Paolo Frasconi
Francesca A. Lisi (E

LNAI 6489

Inductive
Logic Prog

20th International Confere
Florence, Italy, June 2010
Revised Papers

Lecture Notes in Artificial Intelligence 6489
Edited by R. Goebel, J. Siekmann, and W. Wahlster

Subseries of Lecture Notes in Computer Science

Paolo Frasconi   Francesca A. Lisi (Eds.)

Inductive Logic Programming

20th International Conference, ILP 2010
Florence, Italy, June 27-30, 2010
Revised Papers

Springer

Series Editors

Randy Goebel, University of Alberta, Edmonton, Canada
Jörg Siekmann, University of Saarland, Saarbrücken, Germany
Wolfgang Wahlster, DFKI and University of Saarland, Saarbrücken, Germany

Volume Editors

Paolo Frasconi
Università degli Studi di Firenze
Dipartimento di Sistemi e Informatica
Via di Santa Marta, 3, 50139 Firenze, Italy
E-mail: p-f@dsi.unifi.it

Francesca A. Lisi
Università degli Studi di Bari "Aldo Moro"
Dipartimento di Informatica
Campus Universitario "E. Quagliariello"
Via E. Orabona, 4, 70125 Bari, Italy
E-mail: lisi@di.uniba.it

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-21294-9 e-ISBN 978-3-642-21295-6
DOI 10.1007/978-3-642-21295-6
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: Applied for

CR Subject Classification (1998): F.4.1, I.2.3, I.2.6, F.1, H.2.8, H.3, F.3

LNCS Sublibrary: SL 7 – Artificial Intelligence

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume contains a selection of revised papers from the 20th International Conference on Inductive Logic Programming (ILP 2010) held in Firenze, Italy, during June 27–30, 2010.

The ILP conference series started in 1991 and is the premier international forum on logic-based approaches to machine learning. The conference has recently explored several intersections with statistical learning and other probabilistic approaches, expanding research horizons significantly.

The 20th edition was structured with invited talks, regular talks, a poster session, a panel session, and featured for the first time a tutorial day. The invited speakers were Michael Kifer, Avi Pfeffer, and David Poole. Abstracts of their talks can be found in this volume. Gianluigi Greco and Francesco Scarcello presented a tutorial on "Structural Decomposition Methods: Identifying Easy Instances of Hard Problems"; Volker Tresp presented a tutorial on "Multivariate Models for Relational Learning." Ivan Bratko, Luc De Raedt, Peter Flach, Katsumi Inoue, Stephen Muggleton, David Poole, and Ashwin Srinivasan participated in a panel session highlighting successes and future trends of ILP 20 years after the first meeting.

The overall program featured 16 oral presentations and 15 poster presentations. The presentations of both kind were selected on the basis of extended abstracts. Following the recent tradition of the conference, a selection of the papers accepted at ILP 2010 are published in this volume of the *Lecture Notes in Artificial Intelligence* series and in a special issue of the *Machine Learning* journal. From the initially submitted 44 extended abstracts (8 pages in LNCS format), 31 were accepted for presentation at the conference. Each submission was refereed by at least three Program Committee members and was accessible for additional comments by the entire Program Committee (except in the cases of conflict of interest) thanks to the open reviewing model supported by EasyChair. Out of the accepted contributions 5 were selected for the special issue, 11 were published as a long paper (16 pages), and 15 more as a short paper (8 pages) in the proceedings. These papers were prepared after the conference. Supplementary materials for some of the accepted papers can be retrieved from the conference website (`http://ilp2010.dsi.unifi.it/`).

ILP 2010 would not have taken place without the contribution of many people. We would like to thank the invited and tutorial speakers, the panelists, the Program Committee members, the additional reviewers, the authors of submitted papers, the participants, the local organizers (especially Marco Lippi)

as well as the sponsors (the *Artificial Intelligence* journal, the *Machine Learning* journal, the Association for Logic Programming, the University of Bari "Aldo Moro," the PASCAL 2 Network of Excellence, and the Office of Naval Research Global) for their generous financial support.

March 2011

Paolo Frasconi
Francesca Alessandra Lisi

Organization

Program Chairs

Paolo Frasconi	Università degli Studi di Firenze, Italy
Francesca A. Lisi	Università degli Studi di Bari “Aldo Moro”, Italy

Program Committee

Erick Alphonse	Université Paris-Nord, France
Annalisa Appice	Università degli Studi di Bari “Aldo Moro”, Italy
Hendrik Blockeel	Katholieke Universiteit Leuven, Belgium
James Cussens	University of York, UK
Luc De Raedt	Katholieke Universiteit Leuven, Belgium
Saso Dzeroski	Jozef Stefan Institute, Slovenia
Nicola Fanizzi	Università degli Studi di Bari “Aldo Moro”, Italy
Alan Fern	Oregon State University, USA
Peter Flach	University of Bristol, UK
Nuno Fonseca	CRACS-INESC Porto LA, Portugal
Lise Getoor	University of Maryland, USA
Pascal Hitzler	Wright State University, USA
Tamas Horvath	University of Bonn and Fraunhofer AIS, Germany
Katsumi Inoue	NII, Japan
Manfred Jaeger	Aalborg University, Denmark
Kristian Kersting	Fraunhofer IAIS and University of Bonn, Germany
Ross King	University of Wales, UK
Tolga Konik	Stanford University, USA
Stefan Kramer	TU München, Germany
Niels Landwehr	University of Potsdam, Germany
Nada Lavrac	Jozef Stefan Institute, Slovenia
Sofus Macskassy	Fetch Technologies
Donato Malerba	Università degli Studi di Bari “Aldo Moro”, Italy
Lily Mihalkova	University of Maryland, USA
Brian Milch	Google, USA
Stephen Muggleton	Imperial College London, UK
Ramon Otero	University of Corunna, Spain
David Page	University of Wisconsin, USA
Andrea Passerini	Università degli Studi di Trento, Italy
Jan Ramon	Katholieke Universiteit Leuven, Belgium
Oliver Ray	University of Bristol, UK
Chiaki Sakama	Wakayama University, Japan

Vítor Santos Costa	Universidade do Porto, Portugal
Taisuke Sato	Tokyo Institute of Technology, Japan
Jude Shavlik	University of Wisconsin, USA
Takayoshi Shoudai	Kyushu University, Japan
Ashwin Srinivasan	IBM India Research Lab, India
Prasad Tadepalli	Oregon State University, USA
Volker Tresp	Siemens AG Munich, Germany
Christel Vrain	LIFO - Université d'Orléans, France
Stefan Wrobel	University of Bonn and Fraunhofer IAIS, Germany
Akihiro Yamamoto	Kyoto University, Japan
Gerson Zaverucha	PESC/COPPE - UFRJ, Brazil
Filip Zelezny	Czech Technical University, Czech Republic

Additional Reviewers

Aleksovski, Darko
Camacho, Rui
Ceci, Michelangelo
Duboc, Ana Luísa
Dutra, Inês
Gonzalez, Jorge
Hadiji, Fabian
Illobre, Alberto
Kameya, Yoshitaka
Kimmig, Angelika
Lehmann, Jens
Mantadelis, Theofrastos
Nickel, Maximilian
Paes, Aline
Rettinger, Achim
Riguzzi, Fabrizio
Romero, Javier
Slavkov, Ivica
Steinke, Florian
Stojanova, Daniela
Suzuki, Yusuke
Tran, Son

Local Arrangements Committee

Marco Lippi	Università degli Studi di Firenze, Italy
Arianna Sciarrillo	Università degli Studi di Firenze, Italy

Sponsors

Artificial Intelligence journal
Association for Logic Programming
Machine Learning journal
Office of Naval Research Global
PASCAL 2 Network of Excellence
Università degli Studi di Bari "Aldo Moro"

Table of Contents

Rule Interchange Format: Logic Programming's Second Wind?

Michael Kifer

Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794-4400, USA
kifer@cs.stonybrook.edu

Abstract. Recent years have witnessed a strong upswing in the interest in rule systems technologies—both in their own right and in combination with existing Web standards. In particular, the Semantic Web is now seen as a vast playing field for rules within the academia as well as the industry. This renewed interest motivated the development of the Rule Interchange Format (RIF), a recent W3C Web standard for exchanging rules among different and dissimilar systems [1–5]. Despite its name, RIF is not merely a format: it is a collection of concrete rule languages, called RIF dialects, and a framework for defining new ones in harmony with each other. This includes formal specifications of the syntax, semantics, and XML serialization.

In this talk we argue that RIF is a major opportunity to re-introduce rule based technologies into the mainstream of knowledge representation and information processing, and to rekindle the interest in logic programming. First, we will introduce the main principles behind RIF and then discuss the application landscape that could emerge if this standard is embraced by the relevant communities: Logic Programming, Semantic Web, and Knowledge Representation. We will also reflect on the past of logic programming and speculate on how it could benefit from and contribute to RIF in the future.

References

1. Boley, H., Hallmark, G., Kifer, M., Paschke, A., Polleres, A., Reynolds, D.: RIF Core dialect. Working draft, W3C (July 3, 2009)
2. Boley, H., Kifer, M.: RIF Framework for logic dialects. Working draft, W3C (July 3, 2009)
3. Boley, H., Kifer, M.: RIF basic logic dialect. Working Draft, W3C (July 3, 2009)
4. de Sainte Marie, C., Paschke, A., Hallmark, G.: RIF Production rule dialect (March 2009), http://www.w3.org/2005/rules/wiki/PRD
5. Polleres, A., Boley, H., Kifer, M.: RIF Datatypes and built-ins 1.0 (July 2009), http://www.w3.org/2005/rules/wiki/DTB

Practical Probabilistic Programming

Avi Pfeffer

Charles River Analytics, 625 Mount Auburn Street, Cambridge, MA 02140, USA
apfeffer@cra.com

Abstract. Probabilistic programming promises to make probabilistic modeling easier by making it possible to create models using the power of programming languages, and by applying general-purpose algorithms to reason about models. We present a new probabilistic programming language named Figaro that was designed with practicality and usability in mind. Figaro can represent models naturally that have been difficult to represent in other languages, such as probabilistic relational models and models with undirected relationships with arbitrary constraints. An important feature is that the Figaro language and reasoning algorithms are embedded as a library in Scala. We illustrate the use of Figaro through a case study.

Keywords: Probabilistic modeling, representation languages, probabilistic programming.

1 Introduction

Probabilistic models are ever growing in richness and diversity. Models may be hierarchical, relational, spatio-temporal, recursive and infinite, among others. Developing the representation, inference, and learning algorithms for new models is a significant task. Probabilistic programming has the potential to make this task much easier by allowing the modeler to represent rich, complex probabilistic models using the full power of programming languages, including data structures, control mechanisms, functions and abstraction. The language provides inference and learning algorithms so the model designer does not need to implement them. Most importantly, a probabilistic programming language (PPL) provides the modeler with the language and tools with which to think of and formulate new models for complex domains.

To this point, most of the research on PPLs has focused on improving the power of a language, both in terms of what it can represent and in terms of the algorithms it uses to reason about models written in the language. PPLs today can represent an extremely wide range of models and use increasingly sophisticated algorithms. However, there has been less focus hitherto on usability of the languages. Ultimately, the goal of PPL research is to provide languages that can be used by many people, not just experts in probabilistic programming. We present Figaro, a PPL that is designed to be usable without sacrificing any power. In designing the language for usability, we identified the following four goals: (1) Implement the language as a library that can be used by a wide range of programmers; (2) Naturally represent both directed and undirected models with arbitrary constraints; (3) Naturally represent models with interacting objects; (4) Modular, extensible algorithm specification.

An important characteristic of Figaro that distinguishes it from previous probabilistic programming languages is that it is a language not only for *representing* rich probabilistic models but also for *constructing* such models. It achieves this property by being embedded in Scala, which is an object-oriented and functional programming language that is interoperable with Java. Figaro inherits the object-oriented and functional nature of Scala. Languages such as IBAL [1] and Church [2] already demonstrated the power of functional probabilistic programming. As a result of its embedding, a Figaro program can be constructed by an arbitrarily complex Scala program. In particular, the Scala program can create data structures that cannot be achieved by an ordinary functional probabilistic program. Thus, the embedding in Scala achieves the first three goals: Figaro is an extensible library that can be used by many programmers; Figaro can represent undirected models with arbitrary constraints, and even directed cyclic models which have been very difficult to represent to this point; and Figaro can naturally capture object-relational models like probabilistic relational models while maintaining their object structure. The fourth goal is also achieved through the embedding in Scala. Figaro programs are Scala data structures that have declarative semantics as probabilistic models that are independent of any specific reasoning algorithm. Therefore, any reasoning algorithm can be written that respects the semantics and applied to the data structures. Figaro provides an extensible class library of such algorithms.

We illustrate the use of Figaro through a case study, in which we implemented a system to infer the capabilities and intentions of an adversarial agent. Our system was given a history of past situations involving the agent and its goal was to reason about a new situation. Different situations provide the agent with different objectives and means with which to achieve them, but some objectives and means are shared across situations. In addition, different objectives share the same logical structure with different parameterizations, and the realization of that logical structure in a situation depends on which elements are related in that situation. Figaro was able to capture this rich structure easily, using the language to construct a specific model for each situation while sharing knowledge between situations, and exploiting the sharing of logical structure between objectives while relating each objective to the appropriate related objectives.

References

1. Pfeffer, A.: The Design and Implementation of IBAL: A General-Purpose Probabilistic Language. In: Getoor, L., Taskar, B. (eds.) Statistical Relational Learning. MIT Press, Cambridge (2007)
2. Goodman, N.D., Mansinghka, V.K., Roy, D., Bonawitz, K., Tenenbaum, J.B.: Church: A Language for Generative Models. In: Uncertainty in Artificial Intelligence (UAI) (2008)

Probabilistic Relational Learning and Inductive Logic Programming at a Global Scale

David Poole

Department of Computer Science,
University of British Columbia
Vancouver, BC, Canada
http://www.cs.ubc.ca/~poole/

Abstract. Building on advances in statistical-relational AI and the Semantic Web, this talk outlined how to create knowledge, how to evaluate knowledge that has been published, and how to go beyond the sum of human knowledge. If there is some claim of truth, it is reasonable to ask what evidence there is for that claim, and to not believe claims that do not provide evidence. Thus we need to publish data that can provide evidence. Given such data, we can also learn from it. This talk outlines how publishing ontologies, data, and probabilistic hypotheses/theories can let us base beliefs on evidence, and how the resulting world-wide mind can go beyond the aggregation of human knowledge. Much of the world's data is relational, and we want to make probabilistic predictions in order to make rational decisions. Thus probabilistic relational learning and inductive logic programming need to be a foundation of the semantic web. This talk overviewed the technology behind this vision and the considerable technical and social problem that remain.

Keywords: Statistical relational AI, Probabilistic Relational Learning, Semantic Science, Lifted Probabilistic Inference, World-Wide Mind.

To make decisions, we should base them on the best information available; we need to be able to find all relevant information and condition on it effectively. We have called the technology to support decisions "semantic science" [7,8], based on the semantic web, which is an endeavor to make all of the world's knowledge accessible to computers, and using scientific methodology to make predictions. Figure 1 shows the main components of semantic science. Ontologies are used to define the vocabulary of data and hypotheses. These are needed to enable us to find and use all relevant information. Observational data, which depends on the world and the ontologies, are published. Such data sets can be very heterogenous, at widely varying levels of abstraction and detail. Hypotheses that make probabilistic predictions are also published. Hypotheses are not created in isolation, but depend on some training data. Hypotheses can be judged by their prior plausibility and how well they predict the data. Given a new case, various hypotheses are combined to form models that can be used to make predictions on that case. Given a prediction, users can ask what hypotheses were used to make that prediction, and for each hypothesis, users can find the relevant data to evaluate the hypothesis. In this way decisions can be based on all of the applicable evidence.

Typically data is not just a set of mappings from features into a fixed set of values, as is often assumed by traditional machine learning, but often refers to individuals that are only referred to by name; it is the properties of these individuals and the relationships among these individuals that is important for prediction. Following the publication of what could be argued was the first probabilistic relational language [1,2,5], the combination of logical and probabilistic reasoning, and probabilistic programming languages [6] has blossomed. There are still many open fundamental problems for representations, inference and learning. [3] proposed the problem of lifted inference: carrying out probabilistic inference reasoning about classes of individuals as a unit, without reasoning about them individually. Another problem is where models refer to individuals in terms of the roles they fill, but the data does not label the observed individuals with roles [4]. There is still lots of exciting research to be done!

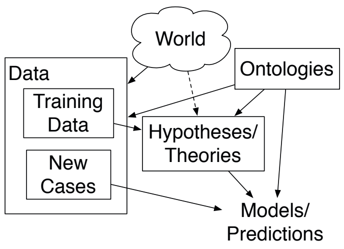


Fig. 1. Semantic Science

References

1. Poole, D.: Probabilistic Horn abduction and Bayesian networks. Artificial Intelligence 64(1), 81–129 (1993)
2. Poole, D.: The independent choice logic for modelling multiple agents under uncertainty. Artificial Intelligence 94, 7–56 (1997); special issue on economic principles of multi-agent systems
3. Poole, D.: First-order probabilistic inference. In: Proc. Eighteenth International Joint Conference on Artificial Intelligence (IJCAI 2003), Acapulco, Mexico, pp. 985–991 (2003)
4. Poole, D.: Logical generative models for probabilistic reasoning about existence, roles and identity. In: 22nd AAAI Conference on AI, AAAI 2007 (July 2007)
5. Poole, D.: The independent choice logic and beyond. In: De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S.H. (eds.) Probabilistic Inductive Logic Programming. LNCS (LNAI), vol. 4911, pp. 222–243. Springer, Heidelberg (2008)
6. Poole, D.: Probabilistic programming languages: Independent choices and deterministic systems. In: Dechter, R., Geffner, H., Halpern, J. (eds.) Heuristics, Probability and Causality: A Tribute to Judea Pearl, pp. 253–269. College Publications (2010)
7. Poole, D., Smyth, C., Sharma, R.: Semantic science: Ontologies, data and probabilistic theories. In: da Costa, P.C.G., d'Amato, C., Fanizzi, N., Laskey, K.B., Laskey, K.J., Lukasiewicz, T., Nickles, M., Pool, M. (eds.) URSW 2005 - 2007. LNCS (LNAI), vol. 5327, pp. 26–40. Springer, Heidelberg (2008)
8. Poole, D., Smyth, C., Sharma, R.: Ontology design for scientific theories that make probabilistic predictions. IEEE Intelligent Systems 24(1), 27–36 (2009)

Learning Multi-class Theories in ILP

Tarek Abudawood and Peter A. Flach

Intelligent Systems Laboratory, University of Bristol, UK
Dawood@cs.bris.ac.uk, Peter.Flach@bristol.ac.uk

Abstract. In this paper we investigate the lack of reliability and consistency of those binary rule learners in ILP that employ the one-vs-rest binarisation technique when dealing with multi-class domains. We show that we can learn a simple, consistent and reliable multi-class theory by combining the rules of the multiple one-vs-rest theories into one rule list or set. We experimentally show that our proposed methods produce coherent and accurate rule models from the rules learned by a well known ILP learner *Aleph*.

1 Introduction

Inductive Logic Programming (ILP) is concerned with inducing first-order clausal models from examples and background knowledge. Symbolic rule learning systems in ILP, such as FOIL [1], PROGOL [2] and Aleph [3], learn rules from positive and negative examples. They are known for their ability to learn from complex structured data and build effective classification models in a range of domains. Unfortunately, they struggle in dealing with multi-class problems. In most situations they reduce a multi-class problem into multiple binary problems following the pairwise one-vs-one or one-vs-rest binarisation techniques.

Aleph, for example, can learn a multi-class theory in the one-vs-rest paradigm where the outcome of its induction can be seen as a combination of several black-box models. Each model induces rules for one specific (positive) class, and a default rule is added to predict the remaining classes. This one-vs-rest approach is a commonly used machine learning technique due to its simplicity in solving multi-class problems. It has been proven to be powerful when compared to other multi-class approaches [4]. However, we argue in this paper that the one-vs-rest technique is not suitable for first-order rule learners as there is a strong bias toward the negative classes leading to unrealistic estimates of predictive power. In addition, the lack of integrity between the different binary models leads to inconsistent predictions.

We investigate the reliability (how much one can rely on the quality of a model) and consistency (how consistent are the predictions of multiple related models) of one-vs-rest binary models and illustrate the difference with a proper multi-class model in Sect. 2. In Sect. 3 our goal is to investigate several methods to overcome the problems of the current application of one-vs-rest technique in ILP rule learners. We experimentally demonstrate the performance of our suggested methods in Sect. 4 and compare them with the standard binary method of Aleph. In Sect. 5 we briefly revisit related work before we draw the conclusion in the final section.

2 Motivation

In machine learning accuracy is commonly used for comparing the classification performance and thus many researchers report their results in terms of accuracy, and compare their results against accuracies of other algorithms. The accuracy of a model can be interpreted as the expectation of correctly classifying a randomly selected example.

		Pred. c^+	Pred. c^-	Tot.
Act.	c^+	TP	FN	E^+
Act.	c^-	FP	TN	E^-
	Tot.	$\hat{E}^+$	$\hat{E}^-$	E

(a)

		Pred. c_1	c_2	c_3	$\dots$	c_n	Tot.
	c_1	TP_1	FN_1	FN_1	$\dots$	FN_1	E_1
	c_2	FN_2	TP_2	FN_2	$\dots$	FN_2	E_2
Act.	c_3	FN_3	FN_3	TP_3	$\dots$	FN_3	E_3
	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
	c_n	FN_n	FN_n	FN_n	$\dots$	TP_n	E_n
	Tot.	$\hat{E}_i$	$\hat{E}_2$	$\hat{E}_3$	$\dots$	$\hat{E}_n$	E

(b)

Fig. 1. Contingency tables for a binary model (a) and a multi-class model (b)

Using the notation explained in Fig. 1, we introduce the following definitions.

Definition 1 (Recall). *The* recall *of a given class c_i, denoted $Recall_i$ or $Recall_i^+$, is the proportion of examples of class c_i that is correctly classified by a model ($Recall_i = TP_i/E_i$). The* negative recall *of class c_i, denoted $Recall_i^-$, is the proportion of examples of class c_i incorrectly classified ($Recall_i^- = 1 - TP_i/E_i$). In case of two classes, positive and negative, we denote the recall of the positive class as $Recall^+ = TP/E^+$ and of the negative class as $Recall^- = TN/E^-$.*

Definition 2 (Accuracy). *Given two classes c^+ and c^-, the* binary accuracy *of a model is defined as*

$$Accuracy^{bin} = \frac{TP+TN}{E} = \frac{E^+}{E}Recall^+ + \frac{E^-}{E}Recall^-$$

That is, binary accuracy is a weighted average of the positive and negative recall, weighted by the class prior. This extends to multiple classes:

$$Accuracy = \sum_{i=1}^{n}\frac{TP_i}{E} = \sum_{i=1}^{n}\frac{E_i}{E}\frac{TP_i}{E_i} = \sum_{i=1}^{n}\frac{E_i}{E}Recall_i^+$$

For this reason we sometimes refer to accuracy as (weighted) average positive recall.

Definition 3 (Multi-Model Accuracy). *Given n classes and n one-vs-rest models, one for each class, the* multi-model accuracy *is defined as the average binary accuracy of the n models:*

$$Accuracy^{mm} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{E_i^+}{E}Recall_i^+ + \frac{E_i^-}{E}Recall_i^-\right)$$

The following simple result is worth noting.

Lemma 1. *The accuracy of a single multi-class model is not equivalent to the multi-model accuracy of the one-vs-rest models derived from the multi-class model.*

Proof.

$$\begin{aligned} Accuracy^{mm} &= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{E_i^+}{E}Recall_i^+ + \frac{E_i^-}{E}Recall_i^-\right) & (1)\\ &= \frac{1}{n}\sum_{i=1}^{n}\frac{E_i^+}{E}Recall_i^+ + \frac{1}{n}\sum_{i=1}^{n}\frac{E_i^-}{E}Recall_i^- & (2)\\ &= \frac{1}{n}Accuracy + \frac{1}{n}\sum_{i=1}^{n}\frac{E_i^-}{E}Recall_i^- & (3)\end{aligned}$$

In going from (2) to (3) we rely on the fact that the one-vs-rest models are derived from a single multi-class model. If this isn't the case (as in Aleph, for instance), then weighted average positive recall is not the same as accuracy, which compounds the issue.

It can be seen from Lemma 1 that the two accuracies are different. Accuracy of a multi-class model relies on the positive recalls weighted by the class priors. On the other hand, the average accuracy of multiple binary models relies on the recalls of both classes where the importance of the positive recalls is decreased n times, hence, there is an increase of the importance of classifying a negative example n times. The following example demonstrates why multi-model accuracy is misleading.

Example 1 (A random classifier). Let us consider a 3-class problem comprising 108 examples uniformly distributed among the classes. A random 3-class classifier would result in the uniform contingency table shown in Fig. 2(a). On the other hand, if a random binary classifier is applied to the three one-vs-rest binary problems we obtain the three contingency tables in Figs. 2(b)-2(d). The accuracy of the multi-class model is 0.33, while the multi-model accuracy of the binary models is 0.50.

		Pred.			
		c_1	c_2	c_3	Tot.
Act.	c_1	12	12	12	36
	c_2	12	12	12	36
	c_3	12	12	12	36
	Tot.	36	36	36	108

(a)

		Pred.		
		c_1	$c_{2,3}$	Tot.
Act.	c_1	18	18	36
	$c_{2,3}$	36	36	72
	Tot.	54	54	108

(b)

		Pred.		
		c_2	$c_{1,3}$	Tot.
Act.	c_2	18	18	36
	$c_{1,3}$	36	36	72
	Tot.	54	54	108

(c)

		Pred.		
		c_3	$c_{1,2}$	Tot.
Act.	c_3	18	18	36
	$c_{1,2}$	36	36	72
	Tot.	54	54	108

(d)

Fig. 2. Four contingency tables on a three-class problem showing the predictions of a random multi-class classifier (a) and three random one-vs-rest classifiers (b)-(d)

It is clear that the average accuracy of the binary models is 1.5 times more than the accuracy of the multi-class model because the weight of the negative class is twice the weight of the positive class. When having a proper multi-class model, there are only credits for classifying examples correctly. Averaging the positive and negative recalls for multiple one-vs-one theories could be misleading but it is even more harmful when it comes to one-vs-rest theories as the problem is propagated.

Another problem arising when inducing multiple independent binary theories is the lack of integrity between the predictions of the different binary theories. This may cause an example to have different possible predictions in several contingency tables because each model produces predictions independently of the others. The predictions of the models on each example should be consistent. For instance, by considering n one-vs-rest models where each model is trained to predict one class as positive, then the prediction for an example x on the i-th model should be be consistent with its prediction on the j-th model, $\hat{c_i}(x) = +ve$ and $\hat{c_j}(x) = -ve \; \forall \; j \neq i$, where $\hat{c_j}(x)$ and $\hat{c_j}(x)$ express the prediction of the i-th and the jth binary model respectively for example x.

If the predictions are inconsistent then such conflicts need to be solved to ensure the consistency in the predictions for each example in all models. All one-vs-rest models of support vector machines and naive Bayes [5] resolve these collisions by obtaining n scores from each one of the n models and the model with the maximum score wins the prediction [4]. A rule learner such as CN2 [6] learns ordered rule lists in one of its settings to avoid such conflicts. In pairwise techniques voting methods [7,8,9,10] can be considered to integrate the predictions.

The discussion about unreliability and inconsistency holds generally when applying one-vs-rest technique in any learning system but we would like to emphasise the importance of this issue particularly in ILP binary rule learning systems such as Aleph. This is because we only induce rules for the positive class in each one-vs-rest model while a default rule that always predicts the negative class is added in case an example can not be classified by any induced rule. The default rules give credits for not classifying negative examples which makes it easy to obtain high negative recalls without inducing any rules for the negatives. For instance, one could obtain 0.67 multi-model accuracy with three empty theories[1] on the problem of Example 1. Hence, there is a need to integrate the different binary models of such rule learning systems in order to ensure the reliability and consistency of their predictions.

3 Improved Learning of Multi-class Theories

In this section we investigate how one could improve the reliability of the all one-vs-rest theories in ILP by combining their binary models into a single rule listor rule set model. Our approach is different from the other first-order rule learning approaches in various respects. First, it does not treat the n various models as independent black-box models, but instead combines the rules of all the models into a single model. Secondly, there is only one default rule and the class of the default rule is determined probabilistically according to the distribution of the uncovered training examples of all the classes. Finally, a single prediction is obtained for each example in one multi-class contingency table. Despite the simplicity of our approaches, their predictions are reliable, consistent and accurate, as we will show in our experiments.

In any rule list model, the rules are ordered in the final theory according to a certain criterion. When an unseen example is encountered, the rules are tried one by one in the order of the list and the first rule that fires determines the class of the example. So the

[1] An empty theory is a theory where a binary rule learner fails to induce any rule for the positive examples.

key idea is how to order these rules. One needs to evaluate the rules induced by the n models and assign them scores. We adopt Chi^2 as our multi-class evaluation measure for the rules and used it to build a Multi-class Rule List (MRL) model.

Definition 4 (Chi-Squared [11]). *The Chi-squared score of a rule r_j of the i-th class is defined as $Chi^2(r_j) = \sum_{i=1}^{n} \frac{[e_i E - eE_i]^2}{eE_i(E-e)}$ where e is the number of examples covered by r_j, e_i the number of examples correctly classified by r_j, E_i is the total number of examples of the i-th class, and E is the total number of examples.*

MRL In this method, after learning rules for all classes, the rules are re-ordered on decreasing Chi^2. The ties are broken randomly. If a rule is added to the rule list, then all examples it covers are removed from the training set and the rest of the rules are re-evaluated based on the remaining examples until no further rule is left. At the end, a single default rule is assigned predicting the majority class of the uncovered examples.

In a rule set model, the rules are unordered and the class of a new example is determined based on the training statistics of all rules that fire for that particular example. For instance, the CN2 rule learner [6] learns a rule set model and tags the rules with their coverage over all the classes. If a new example is to be classified, CN2 sums up the coverage of all rules that fire over each class and the class with the highest coverage wins. We propose two methods to handle multi-class rule set theories, the Multi-class Rule Set Intersection (MRSI) method and the Multi-class Rule Set Union (MRSU) method. The descriptions of the two methods are discussed below.

MRSI In MRSI every rule from the multiple one-vs-rest models is evaluated over the entire training set once, and the identifiers of the examples they cover are stored. A default rule is formed based on the majority class of the uncovered training examples. If a new example is to be classified, all the rules are tried. For those rules that fire, we determine the intersection of their training set coverage using the example identifiers, and their class distribution gives us the empirical probability of each class. The class with the maximum probability is predicted for the example. Again the ties are broken randomly. In the case of an empty intersection, the majority class is assigned to the example.

MRSU The MRSU method differs from the MRSI method in that it determines the class of a new example based on the union of the training coverage of all rules that cover the new example, instead of the intersection.

The MRSU method is closer in spirit to the CN2 method, which adds up the coverage of all rules that fire. However, by using example identifiers we avoid double-counting of examples that are covered by several rules, which means that we obtain proper empirical probabilities rather than CN2's estimates.

4 Empirical Evaluation

In this section we evaluate and compare our proposed single multi-class theory learning methods (MRL, MRSU and MRSI) over 6 multi-class data sets and 5 binary data sets

Table 1. Data sets used in the experiments. The group to the left are multi-class data sets while the group to the right are binary data sets. Starred data sets are propositional; the rest is relational.

Data set no.	Name	Class dist.	Data set no.	Class. dist	
1	Car*	1210, 384, 69, 65	7	Mutagenesis	125, 63
2	Diterpene	447, 355, 352, 155, 71	8	Amine (Alzheimer)	1026, 343
3	Ecoli*	143, 77, 52	9	Choline (Alzheimer)	1026, 343
4	English	50, 50, 50	10	Scopolamine (Alzheimer)	1026, 343
5	Protein	116, 115, 77, 73	11	Toxic (Alzheimer)	1026, 343
6	Scale*	288, 288, 49			

(Table 1). We use Aleph as our base-learner, learning rules for each class in turn. We then turn the rules learned by Aleph into coherent multi-class models using the techniques proposed in this paper. We compare against the CN2 rule set method described above.

For each data set, cross-validated accuracies (Table 2) and AUCs (Table 3) were recorded. MRL method does not produce class probabilities and hence produces a single point in a ROC plot: in this case, AUC boils down to the (unweighted) average of true positive and true negative rates. MRSU, MRSI and CN2 produce class probabilities and hence AUC evaluates their ranking performance in the usual way. A multi-class AUC is obtained by averaging each one-vs-rest AUC weighted by the class prior.

Since averaging performance across data sets has limited meaning as the values may not be commensurate, we report the ranks (1 is best, 4 is worst) of the accuracies and AUCs on each data set. We use the Friedman significance test on these ranks at $p = 0.05$ with Bonferroni-Dunn post-hoc test on our three proposed methods. In the Friedman test we record wins and losses in the form of ranks and ignore the magnitude of these wins and losses. The use of the Friedman test to evaluate multiple classifiers on multiple data sets is considered to be more appropriate than the conventional tests, see [12] for further details. By looking at the average performance rank, and calculating the post-hoc test and Critical Difference ($CD = 1.78/2$) with CN2 as control, on the multi-class data sets, MRSI is significantly better than CN2 on both accuracy and AUC, while MRSU performs significantly worse on AUC. If we take a look at the binary data sets ($CD = 1.95/2$) we can see that both MRL and MRSI are significantly superior over CN2 w.r.t. AUC while no statistical significance is reported regarding their accuracies. The conclusion seems warranted that MRSI is preferable for multi-class data sets, while MRL is preferable for binary data sets.

5 Related Work

As discussed earlier, many ILP rule learning systems including Aleph, PROGOL and FOIL can only induce binary theories and multi-class theories are obtained by converting a multi-class problem into several binary problems. The rules of the final model are, in practice, a combination of independent multiple binary theories. Inductive Logic Constraint (ICL) [13] upgraded the propositional CN2 to handle multi-class first-order theories. Our CN2 implementation is similar to ICL (learning from and handling multi-class structural domains) but was built over Aleph. The experiments demonstrate that we can improve over the CN2 probability estimation method.

Table 2. Accuracies of our new multi-class methods (MRL, MRSU and MRSI) compared against CN2 accuracy, with average ranks in brackets. The 6th column shows the multi-model accuracy as reported by Aleph, which is particularly optimistic for multi-class problems due to overemphasising the default rules. The right-most column shows the average positive recall, which ignores the default rules but is still not equal to multi-class accuracy as conflicting predictions are not taken into account.

	MRL	MRSU	MRSI	CN2	Aleph Standard	
	Multi-class accuracy				Multi-model accuracy	Average recall
1	81.43 (2.00)	81.32 (4.00)	**83.98 (1.00)**	81.38 (3.00)	86.90	82.18
2	83.70 (2.00)	83.55 (3.50)	**84.86 (1.00)**	83.55 (3.50)	91.52	82.91
3	**90.43 (1.00)**	86.77 (4.00)	89.75 (2.00)	88.92 (3.00)	90.27	86.46
4	60.67 (3.00)	58.00 (4.00)	**64.00 (1.00)**	62.67 (2.00)	72.44	48.00
5	80.48 (3.00)	80.69 (2.00)	79.70 (4.00)	**80.94 (1.00)**	89.91	70.82
6	80.64 (2.00)	72.51 (4.00)	**83.68 (1.00)**	76.20 (3.00)	79.04	71.20
Average	79.56 (2.17)	77.14 (3.58)	**80.99 (1.67)**	78.94 (2.58)	85.01	73.59
7	**77.06 (2.00)**	**77.06 (2.00)**	76.55 (4.00)	**77.06 (2.00)**	73.97	73.97
8	60.18 (4.00)	60.91 (3.00)	**65.38 (1.00)**	60.98 (2.00)	77.06	77.06
9	**78.24 (1.00)**	76.07 (3.00)	77.14 (2.00)	75.55 (4.00)	60.11	60.18
10	**76.56 (2.00)**	**76.56 (2.00)**	76.48 (4.00)	**76.56 (2.00)**	76.56	76.56
11	74.95 (3.00)	75.02 (2.00)	74.59 (4.00)	**75.09 (1.00)**	74.80	74.80
Average	73.40 (2.40)	73.12 (2.40)	74.03 (3.00)	**73.05 (2.20)**	72.50	72.51

Table 3. Average one-vs-rest AUCs of our multi-class methods (MRL, MRSU and MRSI) compared against CN2, with average ranks in brackets. The AUCs reported for Aleph are for reference only, as these arise from over-emphasising the default rules.

	MRL	MRSU	MRSI	CN2	Aleph Standard
1	**83.03 (1.00)**	75.21 (3.00)	73.92 (4.00)	75.39 (2.00)	82.80
2	88.72 (4.00)	**89.65 (1.00)**	88.90 (3.00)	89.58 (2.00)	88.66
3	91.97 (3.00)	92.63 (2.00)	**93.38 (1.00)**	91.43 (4.00)	86.78
4	70.50 (4.00)	72.67 (2.00)	**74.15 (1.00)**	72.60 (3.00)	66.33
5	87.28 (2.00)	86.62 (4.00)	**89.03 (1.00)**	86.63 (3.00)	83.29
6	**82.05 (1.00)**	74.03 (3.00)	81.41 (2.00)	73.27 (4.00)	76.38
Average	83.92 (2.50)	81.80 (2.50)	**83.46 (2.00)**	81.48 (3.00)	80.71
7	**64.03 (1.00)**	57.19 (4.00)	57.28 (2.00)	57.19 (3.00)	63.93
8	**60.77 (1.00)**	51.70 (3.00)	57.10 (2.00)	51.39 (4.00)	64.03
9	**74.48 (1.00)**	63.91 (3.00)	72.38 (2.00)	60.93 (4.00)	60.90
10	**55.07 (1.00)**	52.70 (3.50)	52.70 (2.00)	52.70 (3.50)	55.07
11	**65.46 (1.00)**	56.15 (3.00)	57.06 (2.00)	55.53 (4.00)	64.71
Average	**63.96 (1.00)**	56.33 (3.30)	59.30 (2.00)	55.55 (3.70)	61.73

While most of the ILP systems implement the covering approach (separate-and-conquer), TILDE [14] implements a divide-and-conquer approach and induces a single first-order logic multi-class theory that take a form of decision tree. Tree models handle multiple classes naturally. We plan an experimental comparison with TILDE in future work. Several papers suggested different approaches of dealing with multiple binary models [4,15,7,8,9,10,5]. A comparison of many such approaches were made in [4] suggesting a superiority of the one-vs-rest approach in general but they also pointed out that the choice of the binarisation technique makes little difference once we learn good binary models.

6 Concluding Remarks

In this paper we investigated the lack of reliability and consistency of the one-vs-rest technique on multi-class domains. We showed that we could learn a simple and single multi-class rule list (MRLmethod) or rule set (MRSU and MRSI methods) model by combining the rules of all one-vs-rest models and turn them into a coherent multi-class classifier.

Our proposed methods generate consistent and reliable multi-class predictions and we experimentally showed that they produce significant results, w.r.t. accuracy and AUC, on both multi-class and binary domains when compared against the CN2 method.

When classification is made based on rule intersection, MRSI, the best accuracies and AUCs were achieved taking the multi-class data sets into account. Multi-class rule list, MRL, method seem to be suitable for two-class problems. The origin of this difference is subject of ongoing investigations.

References

1. Quinlan, J.R., Cameron-Jones, R.M.: FOIL: A Midterm Report. In: Proc. European Conf. on Machine Learning, pp. 3–20. Springer, Heidelberg (1993)
2. Muggleton, S.: Inverse Entailment and Progol. In: Proc. 6th International Workshop on Inductive Logic Programming, vol. 13, pp. 245–286. Springer, Heidelberg (1995)
3. Srinivasan, A.: The Aleph Manual. Technical report, University of Oxford (2001)
4. Rifkin, R., Klautau, A.: In Defense of One-Vs-All Classification. Machine Learning Research 5, 101–141 (2004)
5. Zadrozny, B., Elkan, C.: Transforming Classifier Scores Into Accurate Multiclass Probability Estimates. In: Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, pp. 694–699. ACM, New York (2002)
6. Clark, P., Niblett, T.: The CN2 Induction Algorithm. Machine Learning 3, 261–283 (1989)
7. Friedman, J.H.: Another approach to polychotomous classification. Technical report, Stanford University, Department of Statistics (1996)
8. Platt, J.C., Cristianini, N.: Large Margin DAGs for Multiclass Classification. In: Advances in Neural Information Processing Systems, vol. 12. MIT Press, Cambridge (2000)
9. Kijsirikul, B., Ussivakul, N., Meknavin, S.: Adaptive Directed Acyclic Graphs for Multiclass Classification. In: Ishizuka, M., Sattar, A. (eds.) PRICAI 2002. LNCS (LNAI), vol. 2417, pp. 158–168. Springer, Heidelberg (2002)
10. Dietterich, T.G., Bakiri, G.: Solving Multiclass Learning Problems via Error-Correcting Output Codes. Artificial Intelligence Research 2, 263–286 (1995)
11. Abudawood, T., Flach, P.: Evaluation Measures for Multi-class Subgroup Discovery. In: Buntine, W., Grobelnik, M., Mladenić, D., Shawe-Taylor, J. (eds.) ECML PKDD 2009. LNCS, vol. 5781, pp. 35–50. Springer, Heidelberg (2009)
12. Demšar, J.: Statistical Comparisons of Classifiers Over Multiple Data Sets. Machine Learning Research 7, 1–30 (2006)
13. De Raedt, L., Van Laer, W.: Inductive Constraint Logic. In: Zeugmann, T., Shinohara, T., Jantke, K.P. (eds.) ALT 1995. LNCS, vol. 997, pp. 80–94. Springer, Heidelberg (1995)
14. Blockeel, H., De Raedt, L.: Top-down Induction of Logical Decision Trees. Artificial Intelligence 101, 285–297 (1997)
15. Hsu, C.-W., Lin, C.-J.: A Comparison of Methods for Multiclass Support Vector Machines. Neural Networks 13, 415–425 (2002)
16. Fawcett, T.: An Introduction to ROC Analysis. Pattern Recognition Letters 27(8), 861–874 (2006)

A Numerical Refinement Operator Based on Multi-Instance Learning

Erick Alphonse[1], Tobias Girschick[2], Fabian Buchwald[2], and Stefan Kramer[2]

[1] Laboratoire d'Informatique de l'université Paris-Nord, 99, Av. Jean-Baptiste Clément, 93430 Villetaneuse, France
[2] Technische Universität München, Institut für Informatik I12, Boltzmannstr. 3, 85748 Garching b. München, Germany

Abstract. We present a numerical refinement operator based on multi-instance learning. In the approach, the task of handling numerical variables in a clause is delegated to statistical multi-instance learning schemes. To each clause, there is an associated multi-instance classification model with the numerical variables of the clause as input. Clauses are built in a greedy manner, where each refinement adds new numerical variables which are used additionally to the numerical variables already known to the multi-instance model. In our experiments, we tested this approach with multi-instance learners available in the Weka workbench (like MI-SVMs). These clauses are used in a boosting approach that can take advantage of the margin information, going beyond standard covering procedures or the discrete boosting of rules, like in SLIPPER. The approach is evaluated on the problem of hexose binding site prediction, a pharmacological application and mutagenicity prediction. In two of the three applications, the task is to find configurations of points with certain properties in 3D space that characterize either a binding site or drug activity: the logical part of the clause constitutes the points with their properties, whereas the multi-instance model constrains the distances among the points. In summary, the new numerical refinement operator is interesting both theoretically as a new synthesis of logical and statistical learning and practically as a new method for characterizing binding sites and pharmacophores in biochemical applications.

1 Introduction and Background

It has often been acknowledged that numerical learning in ILP is limited because of the choice of logic programming as representation language [1,2]. Function symbols are not interpreted in logic programming, they simply are seen as functors of Herbrand terms. For instance, the + function symbol being not interpreted, both terms of the following equation cannot be unified and the equation $X + Y = 0$ cannot be solved. To solve this problem, the hypothesis representation language has been extended by a Constraint Programming Language (CLP) [3]. A large number of CLP languages have been proposed, some with complete and efficient solvers. In ILP, the interpreted predicate symbols are often the same as the ones used in attribute-value learning, like $=, \leq, \geq, \in$, but also linear, non-linear, arithmetic or trigonometric functions have been used [4].

The large family of systems able to learn constraints are all based on the technique introduced in the classical INDUCE system [5] and later popularised and developed in the system REMO [6] and other systems [7,3,1,4,2,8]. This technique separates learning the logical part of the hypothesis from learning its constraint part (usually nominal and numerical constraint variables). If we refer to the covering test definition, for the positive examples, at least one of the possible matching substitutions between the logical part of the hypothesis and the logical part of the positive example must satisfy the constraint part. Conversely, for the negative examples, for all possible substitutions, none must satisfy the constraint part. The key idea is to first compute the set of substitutions matching the hypothesis' logical part with the learning examples, and then from the induced tabular representation, where constraint variables are attributes, learn the constraint part of the hypothesis. Zucker and Ganascia note that such a tabular representation is a multi-instance representation in the general case (the constraints are satisfied by at least one matching substitution to a positive example, and none to a negative example), and that multi-instance learners have to be used to learn the hypothesis' constraint part. The different approaches can be compared with respect to the way they define the hypothesis' logical part and when they delegate learning to an attribute-value or a multi-instance learner. INDUCE completely separates the two processes and first searches for a good logical part (following an lgg-based approach), which is then specialized by constraints. A subsequent approach [6] sets the single logical part beforehand, either user-specified or built from the examples. Model selection can then be used to refine increasingly complex hypotheses. Anthony and Frisch [1] limit the constraint part, only allowing a constraint variable to appear in the clause's head, such that they only deal with a single matching substitution, limiting the interest of delegating numerical learning to attribute-value learners. Other systems [7,4] do not limit the logical part, but also do not use the link between θ-subsumption and multi-instance problems and thus treat all matchings to a positive clausal example as a positive attribute-value example. Recently, Srinivasan *et al.* [8] independently proposed the same approach as Zucker *et al.* [6], where the logical part is restricted to one user-defined clause. Finally, an interesting approach, although not too closely related, is MIR-SAYU [9], which learns drug activity from a multi-instance representation of the drugs. The multi-instance representation does not arise from multiple matchings like in INDUCE, but from the multiple conformations that a drug can take in 3D-space: The authors use a rule-based propositionalisation approach to solve the learning problem and obtain a multi-instance representation by applying it to each conformation.

In this paper, we present an approach that does not limit the logical part of a hypothesis: we search in the hypothesis space for a good logical part which, when introducing constraint variables (presently limited to numerical ones), delegates constraint learning to a multi-instance learner. This is different from the classical INDUCE system and more recent approaches, given that intertwining logical and constraint learning should be able to improve search. This also introduces some interesting properties that can be leveraged by a boosting approach (to be explained below). In the following, we present the technical details of the approach.

2 Method

Before we can describe the method in detail, we have to introduce some notation. Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ denote a training set of classified examples. Each example is described by a set of tuples from several relations over nominal and continuous variables, denoted by x_i, and assigned to a class y_i. We restrict ourselves to binary classification problems in this paper ($y_i \in \{+1, -1\}$). The size of the training set is denoted by $|S| = n$. We follow standard multi-instance terminology and make a distinction between *examples* and *instances*: an example is defined as a bag of instances (to be defined later). As we follow a boosting approach for the outer loop of the algorithm (see below), we have a weight w_i associated with each example, which is initialized to $\frac{1}{n}$.

In the following we will deal with negation-free program clauses. Given a set of clauses, we let t denote the index of the t-th clause C_t. Clauses are learned one after the other, using a generalisation of boosting to real-valued weak hypotheses [10]. The weak learner builds a clause C_t as a hypothesis, where t denotes both the index of the clause and the index of the boosting iteration. The boosting procedure constitutes the outer loop of the algorithm (for details we have to refer to the original publication [10]), whereas the construction of clauses constitutes the inner loop.

Due to the size of the search space, clauses are built in a greedy manner, with one refinement after the other. A refinement consists of the addition of one or several literals to the body of a clause according to the modes of a language bias declaration. The refinement operator providing all specializations of a clause is denoted by $\rho(C)$.

In the following, our starting point is a clause C, which is to be refined in a subsequent step:

$$\begin{aligned} C = \mathit{class}(X, Y) :- \; & p_1(X, X_{1,1}, \ldots, X_{1,n_1}), \\ & \ldots, \\ & p_k(X, X_{k,1}, \ldots, X_{k,n_k}). \end{aligned}$$

Upper-case characters X and Y denote, similar to the definition of the training set S above, the identifier of a clausal example (X) and its class Y (either -1 or $+1$).

Given such a clause, its constraint variables can be obtained by a function

$$\mathit{vars}(C) = \{X_{1,1}, \ldots, X_{1,n_1}, \ldots, X_{k,1}, \ldots, X_{k,n_k}\}.$$

Additionally, we have functions $\mathit{vars}_n(C)$ picking the nominal variables of a clause and $\mathit{vars}_c(C)$ picking the continuous variables ($\mathit{vars}(C) = \mathit{vars}_n(C) \cup \mathit{vars}_c(C)$).

For simplicity and without loss of generality, we assume that exactly one literal is added to clause C in the course of a refinement $C' \in \rho(C)$:

$$\begin{aligned} C' = class(X,Y) :- \; & p_1(X, X_{1,1}, \ldots, X_{1,n_1}), \\ & \ldots, \\ & p_k(X, X_{k,1}, \ldots, X_{k,n_k}), \\ & p_{k+1}(X, X_{k+1,1}, \ldots, X_{k+1,n_{k+1}}) \end{aligned}$$

We also make the assumption that at least one additional continuous variable is available after a refinement. In other words, for each $C' \in \rho(C)$ we assume there exists an $X_{k+1,l} \in vars_c(C')$.

It is clear that due to multiplicities ($1:n$ and $m:n$ relationships between the head variable X and the body variables) multi-instance problems over the body variables arise. As our goal is to improve the capability of ILP learning systems to handle continuous variables, we let the multi-instance problems range only over those variables of a clause. The structure of a clause and the remaining variables only serve to give us the definition of a multi-instance problem. To be more precise, we obtain a dataset for multi-instance learning from first materializing the relation from the body (ranging over all variables $vars(C)$) and subsequently projecting it onto the variables $\{X\} \cup vars_c(C)$.

Proceeding in this way, the question is (a) how to guide the search for suitable clauses and (b) how to decide when to stop refining a clause.
For the former question, we decided to use the *margin* of the classifier (in the sense of boosting). Consider the output of the clause together with the multi-instance classifier is given by a function $h(.)$, which tells us not only the class of the prediction (its sign), but also its confidence. Then the *mean margin* of $h(.)$ can be defined as

$$\bar{\mu}_h = \frac{1}{n} \sum_{i=1}^{n} y_i h(x_i) \tag{1}$$

As to decide when to stop refining a clause, we need a criterion that acts as a regularisation parameter for the multi-instance learner. A natural choice is to limit the number of attributes in the datasets that are passed to it. It translates to limiting the number of constraint variables that can be introduced in the logical part, also regularising its complexity.

For the outer loop generating one clause including a multi-instance classifier after the other, we employ a generalization of AdaBoost to real-valued weak hypotheses [10]. For each example covered by a clause C_t, the function $h_t(.)$ (defined in terms of the clause itself plus its multi-instance classifier $f_t(.)$) will provide a different prediction. For the examples not covered by the clause, the weak hypothesis abstains on them and outputs a prediction of 0. In that sense, this is more general than SLIPPER's rules [11], which either abstain or predict the positive class. The boosting algorithm will focus on those examples in the later stages, forcing the weak learner to search for good logical structures that can discriminate between them:

$$h_t^* = \max_{1 \le i \le n} |h_t(x_i)| \tag{2}$$

$$\mu_t = \frac{1}{h_t^*} \sum_{i=1}^{n} w_{t,i} y_i h_t(x_i) \in [-1, +1] \tag{3}$$

$$w_{t+1,i} \leftarrow w_{t,i} \times (\frac{1 - \mu_t y_i h_t(x_i)/h_t^*}{1 - \mu_t^2}) \tag{4}$$

$$\alpha_t = \frac{1}{2h_t^*} \ln \frac{1 + \mu_t}{1 - \mu_t} \tag{5}$$

Overall, the model that is learned is a sequence of clauses C_t along with associated multi-instance models f_t. Both C_t and f_t give h_t, the weak classifiers that are boosted in the outer loop of the algorithm. Additionally, we have the weights originating from the boosting iterations: $((h_1, \alpha_1), \ldots, (h_T, \alpha_T)) = (((C_1, f_1), \alpha_1), \ldots, ((C_T, f_T), \alpha_T))$. In the following we call the described method *NuRMI* (Numerical Refinement operator based on Multi-Instance learning).

3 Experiments

In this section we give an overview of the datasets we used in this study, describe the experimental setup for our method and the two tested baseline methods. Finally, we compare the results.

3.1 Datasets

First, we describe the datasets that we used in our experimental evaluation of the methods.[1]

Hexose Binding Site Dataset. This dataset was compiled by Nassif *et al.* [12]. It is composed of 80 protein-hexose binding sites (positive set/class) and an equal number of non-hexose binding sites and non-binding surface grooves (negative set/class) selected/extracted from Protein Data Bank [13]. For each molecule we have the 3D coordinates of all atoms that have a distance of less than 10 Å from the binding site center. Additionally, for each atom its charge, hydrogen bonding and hydrophobic properties are provided.

Mutagenicity Dataset. We run our experiments on the (by now classical) regression-friendly mutagenicity dataset introduced by Srinivasan *et al.* [14]. It consists of 188 aromatic and heteroaromatic nitrocompounds with a discretized mutagenicity endpoint (positive or negative log mutagenicity). In our experiments, we use B2 background knowledge which focuses on numerical learning on the partial charges.

[1] We would like to thank David Page for providing the first and third dataset of our study.

Dopamine Agonists Dataset. The dopamine agonists dataset is also provided by Davis *et al.* [15]. It is composed of 23 dopamine agonists with 5 to 50 conformations for each molecule. The discretized activity levels of the dopamine agonists represent the classes. Finally, we have 18 positively labeled instances and 5 negatives. Available features are hydrogen acceptors/donors, hydrophobes and basic nitrogen groups.

3.2 Experimental Setup

As a first comparison, we chose the Aleph (`http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/`) system and the relational decision tree induction algorithm Tilde from the ACE Data Mining System (`http://www.cs.kuleuven.ac.be/~dtai/ACE/`). Both algorithms serve as a baseline for our newly developed algorithm NuRMI. It should be pointed out that neither Aleph nor Tilde was particularly optimized on those datasets. On the other hand, we also did not make any effort to optimize the NuRMI parameters. Therefore, the compared systems should be considered to be at approximately the same level of optimization.

The results for the dopamine dataset are obtained using leave-one-out cross-validation, those for the hexose binding experiments and mutagenicity with ten-fold cross-validation. For the Aleph results on dopamine, we used the parameter settings from Davis *et al.* [15]. The clause length is set to 100, the minimum accuracy of a clause is set to 80 %, and the minimum number of positives examples that a clause must cover is set to five. For results on the hexose binding problem, the parameter settings from the reference paper by Nassif *et al.* [12] were used. The maximal clause length is set to eight literals, with only one in the head. The coverage of maximal five negative training examples is accepted with the objective to cover with a rule as many positive examples as possible and to avoid covering negative examples. Aleph's heuristic search is used to speed up the calculations. In Tilde, we used gain ratio as evaluation function for candidate tests. The remaining parameters are Tilde's default parameters.

3.3 Experimental Results

To investigate the performance of our algorithm compared to the reference classifiers Aleph and Tilde, we tested them on the above datasets.

The predictive accuracies estimated by cross-validation are given in Table 1. JRip, MiSVM and MiSMO are abbreviations of multi-instance learning algorithms implemented in the Weka workbench [16] that were plugged into NuRMI. The results show that on two of the three datasets NuRMI performs favorably.

On the dopamine dataset, we observe that both Aleph and Tilde perform worse than a random-guessing classifier. In contrast, NuRMI outperforms the baseline classifiers and apparently is the best choice on this dataset with a predictive accuracy of 82.61%. This suggests that the coverage measure used by Aleph and the gain ratio splitting criterion used by Tilde to choose tests finally

Table 1. Comparison of NuRMI prediction accuracy (%) with Aleph and Tilde. The result of Progol (*) on the mutagenicity dataset is taken from Srinivasan *et al.* [14]. NuRMI results are given for one, three and five rounds of boosting.

Program	dopamine	hexose	mutagenicity
NuRMI_JRip_1	78.26	53.75	80.34
NuRMI_JRip_3	78.26	51.25	81.39
NuRMI_JRip_5	82.61	51.88	81.46
NuRMI_MiSVM_1	65.22	58.75	66.47
NuRMI_MiSVM_3	73.91	60.63	66.47
NuRMI_MiSVM_5	73.91	61.25	66.47
NuRMI_MiSMO_1	78.26	-	76.13
NuRMI_MiSMO_3	78.26	-	81.92
NuRMI_MiSMO_5	78.26	-	81.82
Aleph	48.00	67.50	-
Tilde	61.29	65.00	79.00
Progol	-	-	79.79*

do not result in rules which are able to generalize sufficiently well on this dataset. On the mutagenicity dataset, only the JRip and MiSMO but not the MiSVM NuRMI approach are able to improve upon the 79.79% reported for Progol. On the hexose binding site data, all three classifiers are above the random guessing baseline of 50%.

We observe that NuRMI, a combination of a coverage-based and a margin-based method, is the most accurate on two of the three datasets tested. However, this experimental comparison has to be extended to shed more light on the relative advantages and disadvantages and also on the trade-offs involved.

4 Conclusion

We presented a novel approach to handling numerical refinements based on multi-instance learning (NuRMI). It aims to combine the strengths of Inductive Logic Programming (interpretability) and statistical learning (predictive power). NuRMI generates clauses in a style typical for ILP systems, and evaluates them based on multi-instance classifiers constructed on the numerical variables of a clause. The confidence of the predictions of multi-instance classifiers also guides search and thus drives the refinement steps. In experiments, we have tested the usefulness of NuRMI for biochemical applications, in particular the characterization of binding sites and pharmacophores, and toxicity prediction (including numerical node labels in graph-like representations of molecules). Although the system still has a lot of unused flexibility, e.g., in the choice of (multi-instance) base learners, our preliminary results already hint at the usefulness in the envisaged application domains.

References

1. Anthony, S., Frisch, A.M.: Generating numerical literals during refinement. In: Lavrač, N., Džeroski, S. (eds.) ILP 1997. LNCS, vol. 1297, pp. 61–76. Springer, Heidelberg (1997)
2. Botta, M., Piola, R.: Refining numerical constants in first order logic theories. Machine Learning 38(1/2), 109–131 (2000)
3. Sebag, M., Rouveirol, C.: Constraint inductive logic programming. In: Advances In Inductive Logic Programming, pp. 277–294 (1996)
4. Srinivasan, A., Camacho, R.: Numerical reasoning with an ILP system capable of lazy evaluation and customised search. Journal of Logic Programming 40(2-3), 185–213 (1999)
5. Dietterich, T.G., Michalski, R.S.: A comparative review of selected methods for learning from examples. In: Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (eds.) Machine Learning, an Artificial Intelligence Approach, vol. 1, pp. 41–81 (1983)
6. Zucker, J.-D., Ganascia, J.-G.: Selective reformulation of examples in concept learning. In: Proc. of ICML 1994, pp. 352–360 (1994)
7. Fensel, D., Zickwolff, M., Wiese, M.: Are substitutions the better examples? Learning complete sets of clauses with Frog. In: Proc. of ILP 1995, pp. 453–474 (1995)
8. Srinivasan, A., Page, D., Camacho, R., King, R.D.: Quantitative pharmacophore models with inductive logic programming. Machine Learning 64(1-3), 65–90 (2006)
9. Davis, J., Costa, V.S., Ray, S., Page, D.: An integrated approach to feature invention and model construction for drug activity prediction. In: Proc. of ICML 2007 (2007)
10. Nock, R., Nielsen, F.: A real generalization of discrete adaboost. In: Brewka, G., Coradeschi, S., Perini, A., Traverso, P. (eds.) Proc. of ECAI 2006, vol. 141 (2006)
11. Cohen, W.W., Singer, Y.: A simple, fast, and effective rule learner. In: Proc. of AAAI 1999, pp. 335–342 (1999)
12. Nassif, H., Hassan, A., Sawsan, K., Keirouz, W., Page, D.: An ILP Approach to Model and Classify Hexose Binding Sites. In: Proc. of ILP 2009, vol. 78 (2009)
13. Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N., Bourne, P.E.: The Protein Data Bank. Nucleic Acids Research 28, 235–242 (2000)
14. Srinivasan, A., Muggleton, S., King, R.D., Sternberg, M.J.E.: Mutagenesis: ILP experiments in a non-determinate biological domain. In: Proc. of ILP 1994, vol. 237, pp. 217–232 (1994)
15. Davis, J., Santos Costa, V., Ray, S., Page, D.: Tightly integrating relational learning and multiple-instance regression for real-valued drug activity prediction. In: Proc. of ICML 2007, vol. 287 (2007)
16. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: An update. SIGKDD Explorations 11 (2009)

Not Far Away from Home: A Relational Distance-Based Approach to Understanding Images of Houses

Laura Antanas, Martijn van Otterlo, José Oramas M., Tinne Tuytelaars, and Luc De Raedt

Katholieke Universiteit Leuven, Belgium

Abstract. Augmenting vision systems with high-level knowledge and reasoning can improve lower-level vision processes, by using richer and more structured information. In this paper we tackle the problem of delimiting conceptual elements of street views based on *spatial relations* between lower-level components, e.g. the element 'house' is composed of windows and a door in a spatial arrangement. We use structured data: each concept can be seen as a graph representing spatial relations between components, e.g. in terms of right, up, close. We employ a distance-based approach between logical interpretations to match parts of images with known examples and provide an experimental evaluation on real images.

1 Introduction

In the context of image interpretation, the field of computer vision has developed many techniques over the past decades for *segmenting*, *classifying* and *recognizing objects* and *scenes*. Many of these techniques use a plethora of low-to medium-level and local features such as *lines*, *blobs*, *regions*, *interest points*, and many more [1, 2], which are mostly employed in feature-based, probabilistic classifiers [3–6]. However, many visual scenes can best be described in terms of *hierarchical structures*, expressing the natural composition of scenes into *objects*, *parts* of objects and lower-level *substructures* [7, 8]. For example, a typical house consists of windows, one or more doors, possibly a chimney; all displayed in a particular *configuration*. Following the hierarchical aspect, the chimney itself is composed of a specific arrangement of local features (e.g. "brick"-like patterns). As a result, we advocate the use of high-level representations such as graphs, and more generally using *logical languages* [9]. The use of such formalisms in vision can improve feature-based approaches. Although high-level representations have been considered before, the actual computational *use* of these languages for representation, inference and learning has been less studied in computer vision (but see [10, 11]).

In this paper, we investigate how logical generalization techniques can help to recognize and delineate substructures in an image. In order to do so, we propose a *distance-based* technique for image interpretation. In a more general framework, our aim is to employ a hierarchical representation of images where each image consists of several layers of information. The base layer is a set of features generated by a vision system, e.g. local patterns. A subsequent layer consists of objects, e.g. windows and doors, and a higher level consists of *configurations* of objects, e.g. houses. In this paper we focus on the delineation of meaningful substructures at one particular layer – the house level.

Fig. 1. (a) Annotated/labeled image (Eindhoven). (b) Annotated image (Eindhoven).

We represent a house as a set of objects and a set of spatial relations defined on them (hence; a relational attribute graph). Each house is annotated with the locations and shapes of windows and doors, such that each image structure is *spatially embedded* in a 2D plane, and objects are related to each other with respect to this space.

Related to our work, several papers have explored structured models for building facades [12, 13], but as far as we know, none of these models is based on distances between logical structures. In [14] a distance measure is employed for images of documents, and similar problems were tackled using inductive rule learning [15]. However, our approach builds on recent general results on distance *metrics* for structured data [17] to show how easily they can be used for computer vision tasks. In addition, we focus on a new problem; that of *delineating* houses in street view images. This can be very useful to enhance GOOGLE Street View images, or for home delivery robots to better localize the destination.

2 Setting

For our problem setting we work with street view images of houses. In the Netherlands many streets exist along which houses are built in one block so as to minimize heat loss and to keep a uniform architecture (Fig. 1). They exhibit some variation in doors and windows appearance (e.g. windows having different frame colors), however often there is considerable *consistency* in the way these elements are *structured* at the house level. A surprisingly limited number of *configurations* define the concept *house*. The precise location, shape or size of a composing elements may vary, but the overall configuration remains intact. For example, the door is always on the left or right side of the house and a window is always above a door. In this work we identify such structures from real images. Utilizing this, we solve the *delineation* problem, where the goal is to distinguish individual houses in images that depict rows of adjacent houses, i.e. of repeated structures. We can use the same setup for other knowledge levels (e.g. from lower-level features to the concept *window*), but in this paper we stop at the level of houses.

Detecting house structures from images assumes access to manually labeled examples of houses. Each house is annotated with the bounding boxes and the labels of its composing elements in the image, i.e windows and doors (Fig. 1(a)). This captures the inherent structure of the concept of a house. The configuration is extracted from these features by defining 2D *spatial relations* such as *right*, *above*, *left*, *below*, *close* and *touch* on the labeled bounding boxes. In this way, any structure can be expressed in terms of bounding boxes, their labels and spatial relations between them.

In order to map images to logical representations, we introduce some terminology. A logical *atom* is an expression of the form $\mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_n)$ where $\mathtt{p}/\mathtt{n}$ is a *predicate symbol* and $\mathtt{t}_i$ are *terms*. We assume a functor-free language, hence terms are built from constants and variables. *Constants* are denoted in lower case and *variables* in upper case. *Ground* atoms do not contain variables and will be called *facts*. A *Herbrand interpretation* i assigns to each fact in the language a truth-value. We identify i with the set of facts $\{\mathtt{a}_1, \ldots, \mathtt{a}_N\}$ to which it assigns *true*. A *substitution* $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$ is an assignment of terms $t_1, \ldots, t_n$ to variables $X_1, \ldots, X_n$. Given i_1 and i_2 interpretations, i_1 *θ-subsumes* i_2 iff there is a substitution θ such that $i_1 \subseteq i_2$.

Now we can describe an image Z as follows. First, we obtain a set of *objects* $\{\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_n\}$, by assigning to each bounding box[1] a constant $\mathbf{o}_i$. We derive a ground atom for each object j in the form $\mathtt{part}(\mathbf{o}_j, \mathtt{label})$, forming the set $\mathbf{O}(Z)$. Second, we use definitions of spatial relations between bounding boxes in our background knowledge (BK) to derive a set of spatial relations that hold among the objects in $\mathbf{O}(Z)$. The resulting set is denoted $\mathbf{R}(Z)$. An example of such an atom derived from the spatial relation *close right* is $\mathtt{cRight}(\mathbf{o}_j, \mathbf{o}_k, \mathtt{distance\_value})$, which says that object $\mathbf{o}_j$ is close[2], on the right of object $\mathbf{o}_k$. The term $\mathtt{distance\_value}$ should be within a threshold for the relation to be true. Similarly, $\mathtt{cAbove}$ and $\mathtt{tRight}$ define *close above* and *touch right*, respectively. The BK can easily be extended, and enables to construct a logical representation of visual data:

Definition 1. *A* ***visual interpretation*** V *of an image* Z *is the union of a set of object atoms* $\mathbf{O}(Z)$ *and the set of spatial relation atoms* $\mathbf{R}(Z)$.

A visual interpretation can be seen as a graph; object atoms are attributed vertices and relation atoms are directed (attributed) edges between the vertices. New relations can be used to extend each visual interpretation, by defining them in the BK or adding new attributes to the existing ones. The key point about visual interpretations is that they are fully determined by the set of objects and the background knowledge. This implies that for one image, we can construct multiple visual interpretations by considering different subsets of the objects in the image, i.e. considering different subgraphs in the image. These graphs – in the context of our application in vision – will typically be connected.

We can now use the visual interpretations as an *instance space*, and in effect, a *concept* represents a set of visual interpretations. For example, some visual interpretations will belong to the concept 'house' whereas many others will not. For a particular image and its corresponding set of objects $\mathbf{o}$, the different possible instances will be the visual interpretations of the subsets $\mathbf{o}' \subseteq \mathbf{o}$. We denote ζ as the set of all labeled examples of a concept, called *prototypes* (Fig. 1(a)).

Example 1. A visual interpretation of the image in Fig. 1(b) is:
$I_{\text{img}} = \{\mathtt{part}(o_1, window), \mathtt{part}(o_2, door), \mathtt{part}(o_3, window), \mathtt{part}(o_4, window),$
$\mathtt{part}(\mathbf{o}_5, door), \mathtt{part}(\mathbf{o}_6, window), \mathtt{part}(\mathbf{o}_7, window), \mathtt{part}(\mathbf{o}_8, window),$
$\mathtt{part}(\mathbf{o}_9, window), \mathtt{part}(\mathbf{o}_{10}, window), \mathtt{cRight}(\mathbf{o}_2, \mathbf{o}_1, 60.0), \mathtt{tRight}(\mathbf{o}_3, \mathbf{o}_2, 1.0),$
$\mathtt{cRight}(\mathbf{o}_4, \mathbf{o}_3, 10.0), \mathtt{cAbove}(\mathbf{o}_9, \mathbf{o}_3, 68.0), \mathtt{cAbove}(\mathbf{o}_{10}, \mathbf{o}_1, 73.0), \mathtt{cRight}(\mathbf{o}_9, \mathbf{o}_{10}, 70.0),$

[1] Bounding box is a general term; in our experiments we employ polygon-like shapes.
[2] In practice we use approximate measures to correct for slight deviations stemming from noise.

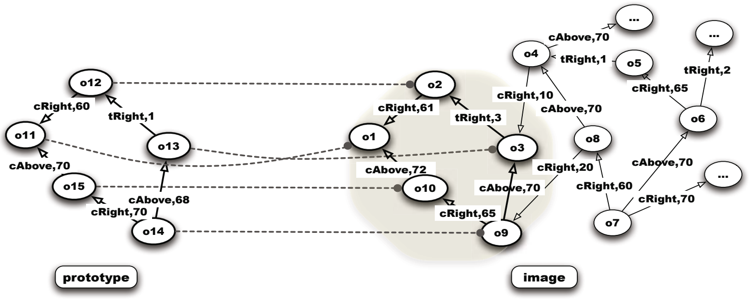


Fig. 2. Graph representations of a prototype and an image interpretation

$\texttt{cRight}(\mathbf{o}_8, \mathbf{o}_9, 20.0), \texttt{cRight}(\mathbf{o}_8, \mathbf{o}_4, 70.0), \texttt{cRight}(\mathbf{o}_7, \mathbf{o}_8, 60.0), \texttt{cAbove}(\mathbf{o}_7, \mathbf{o}_6, 70.0),$ $\texttt{cRight}(\mathbf{o}_6, \mathbf{o}_5, 65.0), \texttt{tRight}(\mathbf{o}_5, \mathbf{o}_4, 1, 0)\}$. The prototype house in Fig. 1(a) is:
$\zeta_i = \{\texttt{part}(\mathbf{o}_{11}, window), \texttt{part}(\mathbf{o}_{12}, door), \texttt{part}(\mathbf{o}_{13}, window), \texttt{part}(\mathbf{o}_{14}, window),$ $\texttt{part}(o_{15}, door), \texttt{cRight}(\mathbf{o}_{12}, \mathbf{o}_{11}, 60.0), \texttt{tRight}(\mathbf{o}_{13}, \mathbf{o}_{12}, 1.0), \texttt{cAbove}(\mathbf{o}_{14}, \mathbf{o}_{13}, 68.0),$ $\texttt{cRight}(\mathbf{o}_{14}, \mathbf{o}_{15}, 70.0), \texttt{cAbove}(\mathbf{o}_{15}, \mathbf{o}_{11}, 68.0)\}$.

Intuitively our goal is to look for known structures in a new image by trying to *embed* prototypes as well as possible in the image. In this direction we define the following:

Definition 2. *A* ***matching*** *between two interpretations i_1 and i_2, $m(i_1, i_2)$, is a mapping such that each atom $\mathbf{a}_1 \in i_1$ corresponds to at most one atom $\mathbf{a}_2 \in i_2$ and vice versa. To each matching we associate a* ***dissimilarity score*** *$d(i_1, i_2)$, which indicates how different the two interpretations are.*

A possible matching between two interpretations (depicted as graphs) is shown in Fig. 2. The quality of the matchings is evaluated by the dissimilarity score. In the next section we will express this score in terms of a *distance metric* between interpretations. We formulate the delineation problem in the following, general, way:

Definition 3. *The* ***delineation problem*** *is defined as:* ***given*** *a set of prototypes ζ, a visual interpretation V of image Z, a dissimilarity score d,* ***find*** *the set of matchings between parts of V and any of the prototypes in ζ,* ***such that*** *all objects appearing in V are matched once, and the score d over the matchings is minimized.*

In effect, solving the delineation problem will carve up the visual interpretation of an image into a set of known structures, i.e. individual houses in this paper.

3 Approach

We propose a possible scoring function d and show how to match prototypes in a new image Z. We combine *structure matching* and *distances* on interpretations. Our method consists of four steps. First, we define spatial BK, and the set of prototypes ζ. Second, we determine all candidate parts of V. Third, we compute distances between all our candidate structures and our prototypes. Fourth, we use the computed distances to find the best delineation. We will now explain the steps.
Step 1 Generate visual interpretations. We first extract image features from Z and generate a set of objects that together with BK forms a visual interpretation V. In ad-

dition, we have a set of labeled prototypes ζ generated in the same manner. We try to find groups of elements which are spatially close and we choose our BK relations accordingly, with relations such as `cRight` (more details in the experimental section).

Step 2 Generate matching candidates. Here we investigate parts of a visual interpretation that could be similar to a prototype. We only select sets of objects (and their corresponding relations) that are *connected*, resulting in the set M. Each element m of M is a possible candidate matching, and m must consist of at least two object atoms and a relation atom and at most all atoms in V. To be able to find the best delineation in case of noisy information, candidates with a small number of atoms are also needed. For example, if the image contains only a part of a (hypothetical) house, containing for example a door or window, they could be grouped with other elements or can be regarded as configurations on their own to best fit the image.

Step 3 Compute distances. To compute the quality of a matching we use a distance metric between two visual interpretations. It evaluates how well the two interpretations match structurally. Although other solutions exist ([16]), here we employ a recent result of [17] which shows that one can construct a metric for any partially ordered hypothesis space L (such as a subsumption lattice) under some mild assumptions. That is, d is a metric if it is defined in terms of the generality order on the hypothesis space, $|.|$ an (anti-monotonic) and strict order preserving *size function* and $d(x,y) = |x| + |y| - 2|\operatorname{mgg}(x,y)|, \forall x, y \in L$. Here, mgg denotes the *minimally general generalization* of two hypotheses x and y. The result allows to derive distance metrics for different types of objects, including graphs, and therefore this result can be used to compute distances between interpretations. For example, one can choose a *graph isomorphism* as a partially ordered relation which induces a generality order on graphs, where the size function can be the number of vertices (see also [18]). Here the mgg corresponds to the *maximal common subgraph*. Computing the distance between graphs g_1 and g_2 is equivalent to calculating the distance between their corresponding visual interpretations using $\operatorname{mgg}(g_1, g_2)$.

Compared to the *least general generalization* (lgg), which is obtained under θ-subsumption, the mgg represents computing the lgg under the assumption of *object identity* (OI) [19]. The lgg could also be used to find a *common part* between interpretations (resp. graphs) but it allows for different variables in the lgg to unify. This collapse of literals into one would violate the strictly ordering preserving condition for the size function. The mgg on the other hand is not unique (we can find multiple common parts) and is the result of exact *structure matching*, i.e. each constant in an interpretation (resp. each node in a graph) must be matched against a *different* constant (resp. node) in the other. Exact structure matching makes also more sense in our setting, since we want to find specific structures and do not want for example to collapse two windows into one. An illustration of mgg under OI-assumption is shown in Example 2.

Example 2. Let $i_1 = \{\mathtt{cRight}(\mathtt{o}_1, \mathtt{o}_2, 2)\}$ and $i_2 = \{\mathtt{cRight}(\mathtt{o}_3, \mathtt{o}_4, 2), \mathtt{cRight}(\mathtt{o}_5, \mathtt{o}_4, 2)\}$. Under θ-subsumption $\operatorname{mgg}_\theta = \{\mathtt{cRight}(\mathtt{X}_1, \mathtt{X}_2, 2), \mathtt{cRight}(\mathtt{X}_3, \mathtt{X}_2, 2)\}$ with $\theta_1 = \{\mathtt{X}_1/\mathtt{o}_1, \mathtt{X}_2/\mathtt{o}_2, \mathtt{X}_3/\mathtt{o}_1\}$, $\theta_2 = \{\mathtt{X}_1/\mathtt{o}_3, \mathtt{X}_2/\mathtt{o}_4, \mathtt{X}_3/\mathtt{o}_5\}$. Under OI-subsumption there are two possible mggs:

$\operatorname{mgg}^0_{OI} = \{\mathtt{cRight}(\mathtt{X}_1, \mathtt{X}_2, 2)\}$ with $\theta^0_1 = \{\mathtt{X}_1/\mathtt{o}_1, \mathtt{X}_2/\mathtt{o}_2\}$, $\theta^0_2 = \{\mathtt{X}_1/\mathtt{o}_3, \mathtt{X}_2/\mathtt{o}_4\}$ and
$\operatorname{mgg}^1_{OI} = \{\mathtt{cRight}(\mathtt{X}_1, \mathtt{X}_2, 2)\}$ with $\theta^1_1 = \{\mathtt{X}_1/\mathtt{o}_1, \mathtt{X}_2/\mathtt{o}_2\}$, $\theta^1_2 = \{\mathtt{X}_1/\mathtt{o}_5, \mathtt{X}_2/\mathtt{o}_4\}$.

Algorithm 1. Step 4 **Delineate the image**

```
Require: prototypes $\zeta$, distance function $d$, visual interpretation $V$ and matchings $M$
1: compute $D_i = \sum_j \frac{d_{ij}}{|\zeta|}, \forall m_i \in M$
2: rank $m_i \in M$ according to $D_i$, select the $k$-best, forming $M_k = \{(m_i, D_i)\}$
3: let $S$ be all subsets of $M_k$ such that $\forall S' \in S$ all object atoms in $V$ appear exactly once in $S'$
4: rank all full solutions $S' \in S$ according to $d_s = \sum_{(m_i,D_i)\in S'} D_i$
5: return the $n$-best solutions $S^*_n = \{S'\}$
```

Given the set $\mathrm{mgg}_{\mathrm{all}} = \{\mathrm{mgg}(i_1, i_2)\}$, where i_1 and i_2 are interpretations we now define the distance between them in the sense of structural matching as:

$$d(i_1, i_2) = \min_{m \in \mathrm{mgg}_{\mathrm{all}}} (|i_1| + |i_2| - 2|m|) \tag{1}$$

where $|\ .\ |$ is the number of atoms in the interpretation or in the mgg. In practice, we use a normalized metric $d(i_1, i_2) = \min_{m \in \mathrm{mgg}_{\mathrm{all}}} (1 - |m|/max(|i_1|, |i_2|))$. Now we can calculate $d_{ij} = d(m_i, \zeta_j)$ which is the distance between matching m_i and prototype ζ_j.
Step 4 Delineate the image. We use d_{ij} to find the best delineation (see Algorithm 1). D_i is the average of the distances from the candidate matching m_i to all the prototypes of a same concept[3]. Hence, it deals with the situation when among prototypes there are noisy examples[4]. We select the first k pairs (m_i, D_i) with the smallest distance, obtaining the set M_k. In a third step we perform a complete search among all subsets of M_k and select the subsets S satisfying certain constraints. In this work we enforce that the union of all object atoms in a subset $S' \in S$ is the set of object atoms in the image interpretation V and the union of all relation atoms in S' is included or equal to the set of relation atoms in V. A possible variation is to allow more relaxed versions of delineations where only some parts of the image are matched, or where matchings can overlap. The constraints can then enforce that e.g. a tree cannot be part of a house. Finally, we select from S the set $S^*_n = \{S'\}$, where S' is among the n solutions that best minimize the sum of distances d_s. Once the delineation at the house level is obtained we can use this information at a next layer (e.g. streets).

4 Experimental Setup and Results

For our experiments we use street view images from Eindhoven. In our dataset there are two possible configurations depending on the position of the door (on the right or left side of the house, see Fig. 1(a)). The image dataset was collected using GOOGLE Street View, and we used the MATLAB toolbox for the LABELME image database [20] to annotate our images. For each image we annotated the windows and the doors. For the training images we annotated also the houses (Fig. 1(a)).

The data is represented in XML format and then translated into PROLOG format. We use *close to the right* (`cRight`), *close above* (`cAbove`) and *touch to the right* (`tRight`) as spatial relations. Thresholds are used on the distance between house elements for

[3] In our case the concept of house.

[4] This can happen when perfect examples are not available, but variations from prototypes are.

Fig. 3. No complete occlusions: (a) Correct delineation. (b) Delineation obtained.

Fig. 4. Image with 5 occluded elements: (a) Correct delineation. (b) Delineation obtained.

Table 1. Delineation results. The accuracy increases as more top ranked solutions are considered.

n first solutions	1	2	3	4	5	6	7	8	9	10
Accuracy	0.73	0.76	0.83	0.9	0.93	0.93	0.93	0.93	0.93	0.96

`close` ($10 \leq \theta \leq \theta_{max}$) and `touch` ($\theta < 10$). θ_{max} is defined relatively to the size of the objects in the image. The amount k of best matchings is heuristically set to $min(200 + 20\% \cdot |M|, 500)$. Choosing a k too small leads to finding no solution, while a high value can give a large search space, prevented by up-bounding k. However, the best solutions are likely to be found among the first ranked candidates. We use 2 noise-free house structures (Fig. 1(a)), one for each configuration, to delineate 30 new test images of houses with the same characteristics as the prototypes, but also different in appearance, size of house elements and distances between elements. Yet, they keep the same repeated structure of the houses as in the examples and also contain several occluded elements. We are able to delineate the houses for less occluded (Fig. 3) and for noisier images (Fig. 4). For all images we considered the first $1 \leq n \leq 10$ solutions, based on the distance value d_s. Table 1 shows the experimental evaluation in terms of accuracy, i.e. the percentage of images with a correct delineation. A delineation of an image is correct if all individual houses in that image are recognized.

5 Conclusions

In this paper we have introduced a simple technique in which logic and distances between relational interpretations are used for the recognition of known structures in images. We have shown that our algorithm can identify substructures that form individual houses, effectively delineating a block of houses. Both the delineation problem, as well as the logical decomposition utilizing distances are relatively novel aspects of our approach. Future work includes incorporating attribute values in the distance function, richer background knowledge bases, and more complex house structures. The most prominent direction is that of replicating our approach in a hierarchical fashion, thereby performing the interpretation process from low-level to high-level. Another, straightforward direction is to employ first-order kernels [21] as our distance function.

References

1. Forsyth, D.A., Ponce, J.: Computer Vision: A Modern Approach. Prentice-Hall, Englewood Cliffs (2003)
2. Tuytelaars, T., Mikolajczyk, K.: Local invariant feature detectors: A survey. Foundations and Trends in Computer Graphics and Vision 3(3), 177–280 (2007)
3. Li, L.-J., Socher, R., Fei-Fei, L.: Towards total scene understanding: Classification, annotation and segmentation in an automatic framework. In: IEEE Conference on Computer Vision and Pattern Recognition (2009)
4. Wang, G., Zhang, Y., Li, F.: Using dependent regions for object categorization in a generative framework. In: Proceedings of CVPR 2006, pp. 1597–1604 (2006)
5. Sudderth, E.B., Torralba, A., Freeman, W.T., Willsky, A.S.: Describing visual scenes using transformed objects and parts. Int. J. on Computer Vision 77(1-3), 291–330 (2008)
6. Bar-Hillel, A., Weinshall, D.: Efficient learning of relational object class models. IJCV 77(1-3), 175–198 (2008)
7. Witkin, A., Tenenbaum, J.: On the role of structure in vision. In: Beck, J., Hope, B., Rosenfeld, A. (eds.) Human and Machine Vision. Academic Press, New York (1983)
8. Pinz, A.J., Bischof, H., Kropatsch, W.G., Schweighofer, G., Haxhimusa, Y., Opelt, A., Ion, A.: Representations for cognitive vision: A review of appearance-based, spatio-temporal, and graph-based approaches. Elec. Let. on Comp. Vision and Im. Analysis 7(2) (2008)
9. De Raedt, L.: Logical and Relational Learning. Springer, Heidelberg (2008)
10. Needham, C.J., Santos, P.E., Magee, D.R., Devin, V.E., Hogg, D.C., Cohn, A.G.: Protocols from perceptual observations. AI 167(1-2), 103–136 (2005)
11. Tran, S.D., Davis, L.S.: Event modeling and recognition using markov logic networks. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part II. LNCS, vol. 5303, pp. 610–623. Springer, Heidelberg (2008)
12. Hartz, J., Neumann, B.: Learning a knowledge base of ontological concepts for high-level scene interpretation. In: ICMLA 2007, pp. 436–443 (2007)
13. Müller, P., Zeng, G., Wonka, P., Van Gool, L.J.: Image-based procedural modeling of facades. ACM Transactions on Graphics 26(3), 85 (2007)
14. Esposito, F., Malerba, D., Semeraro, G.: Classification in noisy environments using a distance measure between structural symbolic descriptions. IEEE TPAMI 14(3), 390–402 (1992)
15. Esposito, F., Ferilli, S., Basile, T.M.A., Mauro, N.D.: Discovering logical structures in digital documents. In: Intelligent Information Systems, pp. 513–521 (2004)
16. Ramon, J., Bruynooghe, M.: A framework for defining distances between first-order logic objects. In: Page, D.L. (ed.) ILP 1998. LNCS, vol. 1446, pp. 271–280. Springer, Heidelberg (1998)
17. De Raedt, L., Ramon, J.: Deriving distance metrics from generality relations. Pattern Recognition Letters 30(3), 187–191 (2009)
18. Bunke, H., Shearer, K.: A graph distance metric based on the maximal common subgraph. Pattern Recognition Letters 19(3-4), 255–259 (1998)
19. Khoshafian, S., Copeland, G.: Object identity. In: 1st ACM OOPSLA, pp. 406–416 (1986)
20. Russell, B.C., Torralba, A., Murphy, K.P., Freeman, W.T.: LabelMe: A database and web-based tool for image annotation. Int. J. of Computer Vision 77(1-3), 157–173 (2008)
21. Gärtner, T.: A survey of kernels for structured data. SIGKDD Explorations 5, 49–58 (2003)

Approximate Inference for Logic Programs with Annotated Disjunctions

Stefano Bragaglia and Fabrizio Riguzzi

DEIS – University of Bologna, ENDIF – University of Ferrara
{stefano.bragaglia,fabrizio.riguzzi}@unibo.it

Abstract. Logic Programs with Annotated Disjunctions (LPADs) are a promising language for Probabilistic Inductive Logic Programming. In order to develop efficient learning systems for LPADs, it is fundamental to have high-performing inference algorithms. The existing approaches take too long or fail for large problems. In this paper we adapt to LPAD the approaches for approximate inference that have been developed for ProbLog, namely k-best and Monte Carlo.

k-Best finds a lower bound of the probability of a query by identifying the k most probable explanations while Monte Carlo estimates the probability by smartly sampling the space of programs. The two techniques have been implemented in the `cplint` suite and have been tested on real and artificial datasets representing graphs. The results show that both algorithms are able to solve larger problems often in less time than the exact algorithm.

Keywords: Probabilistic Inductive Logic Programming, Logic Programs with Annotated Disjunctions, ProbLog.

1 Introduction

Statistical Relational Learning and Probabilistic Inductive Logic Programming provide successful techniques for learning from real world data. Such techniques usually require the execution of a high number of inferences in probabilistic logics, which are costly tasks. In order to reduce the computational load, we may resort to approximate inference that trades accuracy for speed. In this paper we present two approaches for computing the probability of queries from Logic Programs with Annotated Disjunctions (LPADs) [6] in an approximate way. LPADs are particularly interesting because of their sound semantics, of their intuitive syntax and because they allow to exploit many of the techniques developed in Logic Programming for probabilistic reasoning. We present two approaches inspired by those available for ProbLog [2]: k-best and Monte Carlo. The first finds a lower bound for the probability of a query by considering only the k most probable explanations, while the latter estimates the probability of the query by the fraction of sampled possible worlds where the query is true.

2 Logic Programs with Annotated Disjunctions

A Logic Programs with Annotated Disjunctions T [6] consists of a finite set of disjunctive clauses of the form $(H_1 : \alpha_1) \vee (H_2 : \alpha_2) \vee \ldots \vee (H_n : \alpha_n) \leftarrow B_1, B_2, \ldots B_m$ called *annotated disjunctive clauses.* The H_i, B_i and α_i that appear in such a clause are respectively logical atoms, logical literals and real numbers in the interval $[0, 1]$ such that $\sum_{i=1}^{n} \alpha_i \leq 1$. If $\sum_{i=1}^{n} \alpha_i < 1$, the head of the annotated disjunctive clause implicitly contains an extra atom *null* that does not appear in the body of any clause and whose annotation is $1 - \sum_{i=1}^{n} \alpha_i$. For a clause C of the form above, we define $head(C)$ as $\{(H_i : \alpha_i)|1 \leq i \leq n\}$ if $\sum_{i=1}^{n} \alpha_i = 1$ and as $\{(H_i : \alpha_i)|1 \leq i \leq n\} \cup \{(null : 1 - \sum_{i=1}^{n} \alpha_i)\}$ otherwise. Moreover, we define $body(C)$ as $\{B_i|1 \leq i \leq m\}$, $H_i(C)$ as H_i and $\alpha_i(C)$ as α_i.

In order to define the semantics of an LPAD T, we need to consider its grounding $ground(T)$ that must be finite, so T must not contain function symbols if it contains variables. More specifically, an *atomic choice* is a triple (C, θ, i) where $C \in T$, θ is a substitution for the variables of C and $i \in \{1, \ldots, |head(C)|\}$ meaning that the head $H_i(C)\theta : \alpha_i(C)$ was chosen for the clause $C\theta$. A *composite choice* κ is a set of atomic choices that are ground ($C\theta$ is ground) and consistent ($(C, \theta, i) \in \kappa, (C, \theta, j) \in \kappa \Rightarrow i = j$, meaning that only one head is selected for a ground clause) whose probability $P(\kappa)$ is given by $P(\kappa) = \prod_{(C,\theta,i)\in\kappa} \alpha_i(C)$. A *selection* σ is a composite choice containing an atomic choice (C, θ, i) in σ for each clause $C\theta$ in $ground(T)$ and identifies a normal logic program w_σ called a *possible world* (or simply *world*) of T and defined as follows $w_\sigma = \{(H_i(C)\theta \leftarrow body(C))\theta|(C, \theta, i) \in \sigma\}$.

$\mathcal{W}_T$ denotes the set of all the possible worlds of T. Since selections are composite choices, we can assign a probability to possible worlds: $P(w_\sigma) = P(\sigma) = \prod_{(C,\theta,i)\in\sigma} \alpha_i(C)$. The probability of a closed formula ϕ according to an LPAD T is given by the sum of the probabilities of the possible worlds where the formula is true according to the WFS: $P(\phi) = \sum_{\sigma\in\mathcal{W}_T, w_\sigma\models\phi} P(\sigma)$. It is easy to see that P satisfies the axioms of probability.

In order to compute the probability of a query from a probabilistic logic program, [6] proposed to first find a covering set of explanations for the query and then compute the probability from the set by using Binary Decision Diagrams. An *explanation* is a composite choice κ such that the query is true in all the possible worlds consistent with κ. A set K of explanations is *covering* if each possible world where the query is true is consistent with at least one of the explanations in K.

The `cplint` system[1] [5] applied this approach to LPADs. `cplint` first computes a covering set of explanations for a query by using a Prolog meta-interpreter that performs resolution and keeps a set of atomic choices that represents a partial explanation. Each time the meta-interpreter resolves the selected goal with a disjunctive clause, it adds a (possibly non-ground) atomic choice to the partial explanation and checks for its consistency. If the program is range-restricted, when the meta-interpreter reaches the empty goal, every atomic choice in the

[1] `http://www.ing.unife.it/software/cplint/`

partial explanation becomes ground and an explanation is obtained. By enclosing the meta-interpreter in a `findall` call, a covering set K of explanations is found. Then `cplint` converts K into the following Disjunctive Normal Form (DNF) logical formula $F = \bigvee_{\kappa \in K} \bigwedge_{(C,\theta,i) \in \kappa} (X_{C\theta} = i)$. The probability of the query is then given by the probability of F taking value 1. F is converted to a Decision Diagram that is traversed by using a dynamic programming algorithm to compute the probability. Specifically, `cplint` uses Binary Decision Diagram (BDD) because of the availability of highly efficient packages for processing them. Since disjunctive clauses may contain any number of logical heads, multivalued variables are binary encoded by means of boolean variables to be used in BDDs.

3 Approximate Inference

In some domains, computing exactly the probability of a query may be impractical and it may be necessary to resort to some forms of approximations. [2,3] proposed various approaches for approximate inference. With *iterative deepening*, upper and lower bounds for the probability of the query are computed and their difference is gradually decreased by increasing the portion of the search tree that is explored. With the *k-best* algorithm, only the k most probable explanations are considered and a lower bound is found. With *Monte Carlo*, the possible worlds are sampled and the query is tested in the samples. An estimate of the probability of the query is given by the fraction of sampled worlds where the query succeeds. All three approaches have been adapted to LPADs and included in `cplint`. In the following we report only on the k-best and Monte Carlo, since iterative deepening was not giving clear advantages with respect to exact inference on the datasets tested.

3.1 k-best Algorithm

According to [3], using a fixed number of proofs to approximate the probability is fundamental when many queries have to be evaluated because it allows to control the overall complexity. The k-best algorithm uses branch and bound to find the k most probable explanations, where k is a user-defined parameter. The algorithm records the k best explanations. Given a partial explanation, its probability (obtained by multiplying the probability of each atomic choice it contains) is an upper bound on the probability that a complete explanation extending it can achieve. Therefore, a partial explanation can be pruned if its probability falls below the probability of the k-th best explanation. Our implementation of the k-best algorithm interleaves tree expansion and pruning: a set of partial explanations are kept and are iteratively expanded for some steps. Those whose upper bound is worse than the k-th best explanation are pruned. Once the proof tree has been completely expanded, the k best explanations are translated into a BDD to compute a lower bound of the probability of the query. This solution uses a meta-interpreter while ProbLog uses a form of iterative deepening that builds derivations up to a certain probability threshold and then increases the

Algorithm 1. Function SOLVE

```
function SOLVE($Goal, Explan$)
    if $Goal$ is empty then
        return 1
    else
        Let $Goal = [G|Tail]$
        if $G =(\backslash + \ Atom)$ then
            $Valid :=$solve($[Atom], Explan$)
            if $Valid = 0$ then
                return solve($Tail, Explan$)
            else
                return 0
            end if
        else
            Let $L$ be the list of couples $(GL, Step)$ where $GL$ is obtained by resolving
            $Goal$ on $G$ with a program clause $C$ on head $i$ with substitution $\theta$
            and $Step = (C, \theta, i)$
            return SAMPLE_CYCLE($L, Explan$)
        end if
    end if
end function
```

threshold if k explanations have not been found. The meta-interpreter approach has the advantages of avoiding to repeat resolution steps at the expense of a more complex bookkeeping.

3.2 Monte Carlo Algorithm

In [3] the Monte Carlo algorithm for ProbLog is realized by using a vector with an entry for every probabilistic fact. The entries store whether the facts have been sampled true, sampled false or not yet sampled. The vector is initialized with not yet sampled for all facts. Then a transformed ProbLog program is executed that derives the goal and updates the vector each time a new probabilistic fact is sampled.

ProbLog's algorithm requires all the probabilistic facts to be ground in the input program. While LPADs can be converted to ProbLog programs [1], the result of the conversion may contain non ground probabilistic facts so ProbLog's Monte Carlo algorithm may not always be used.

Our Monte Carlo algorithm for LPADs uses a meta-interpreter that keeps a partial explanation containing atomic choices for the disjunctive clauses sampled up to that point. The meta-interpreter is realized by Function SOLVE in Algorithm 1 and returns 1 if the list of atoms of the goal is derivable in the sample and 0 otherwise. In order to derive the selected literal G of the current goal, SOLVE finds all the matching clauses and builds a list of couples (new goal, atomic choice) for each matching clause. Then, it calls Function SAMPLE_CYCLE in Algorithm 2 whose aim is to perform sampling steps for the matching clauses until the truth of the selected literal is determined and a consistent set of ground

Algorithm 2. Function SAMPLE_CYCLE

```
 1: function SAMPLE_CYCLE($L, Explan$)
 2:     $Derivable = 0$
 3:     while $Derivable = 0$ and $L \neq \emptyset$ do
 4:         Remove the first element $(GL, (C, \theta, i))$ from $L$
 5:         repeat
 6:             if $C\theta$ is ground then
 7:                 if $(C, \theta)$ is already present in $Explan$ with head $j$ then
 8:                     $h := j$
 9:                 else
10:                     $h :=$SAMPLE($C$)
11:                 end if
12:             else
13:                 $h :=$SAMPLE($C$)
14:             end if
15:             $Explan := Explan \cup \{(C, \theta, h)\}$
16:             if $h = i$ then
17:                 $Derivable :=$SOLVE($GL, Explan$)
18:             else
19:                 $Derivable := 0$
20:             end if
21:         until CONSISTENT($Explan$)
22:     end while
23:     return $Derivable$
24: end function
```

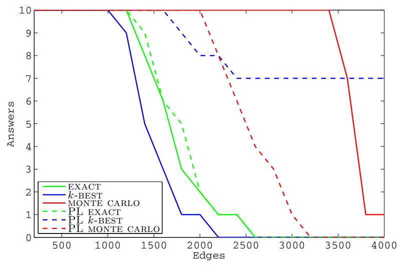
atomic choices is obtained. Each matching clause is sampled independently and the resulting atomic choice is added to the partial explanation that is passed by reference to future calls of SOLVE and SAMPLE_CYCLE.

Since a matching clause may be sampled when it is still not completely ground, further grounding/sampling may lead to inconsistency in the partial explanation. To address this problem, sampling is repeated until a consistent partial explanation is found. The algorithm is guaranteed to terminate because the same head will be eventually sampled for each couple of identical groundings of a clause. Also note that the sampling distribution is not affected since inconsistency arises independently of the success or failure of a query.

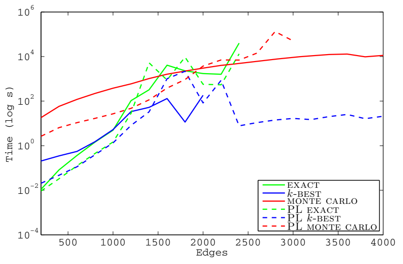
In Algorithm 2, CONSISTENT($Explan$) returns true if $Explan$ is consistent while SAMPLE(C) samples a head index for clause C. SOLVE is called repeatedly to obtain the samples of truth values for the goal. The fraction of true values is an estimation of the probability of the query of interest. The confidence interval on those samples is computed every m samples and the simulation ends when its value drops below a user-defined δ.

4 Experiments

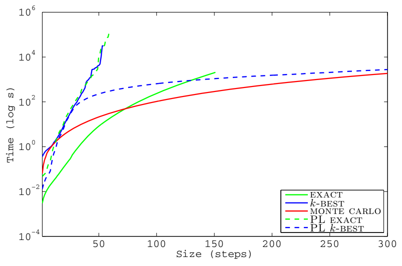
We considered three datasets: graphs of biological concepts from [2], artificial graphs and the UWCSE dataset from [4]. All the experiments have been



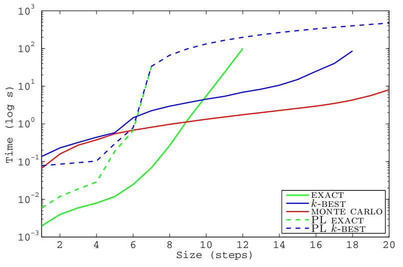
(a) Successes on biological graphs.



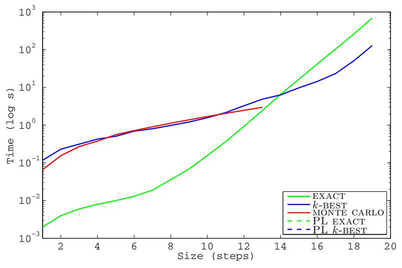
(b) Execution times on biological graphs.



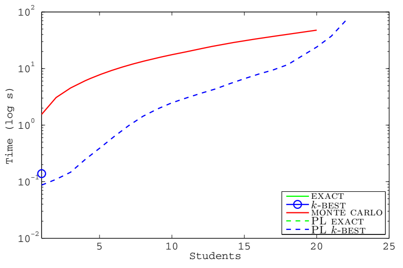
(c) Execution times on Lanes graphs.



(d) Execution times on Branches graphs.



(e) Execution times on Parachutes graphs.



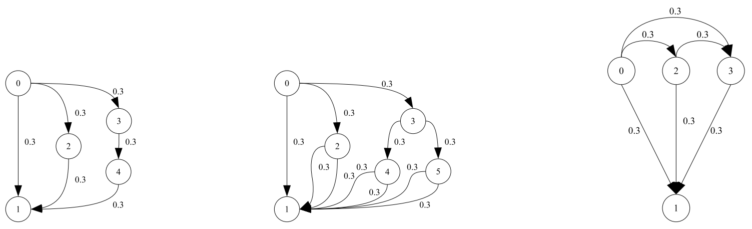
(f) Execution times on UWCSE graphs.

Fig. 1. Experimental results

performed on Linux machines with an Intel Core 2 Duo E6550 (2333 MHz) processor and 4 GB of RAM. The algorithms were implemented in YAP Prolog and run on the data for 24 hours or until the program ended for lack of memory. The values used for the parameters are $k = 64$ as the number of explanations to consider for k-best and $\delta = 0.01$ as the maximum confidence interval width for Monte Carlo algorithm because they represent a good compromise between speed and accuracy.

The biological networks represent relationships among biological entities. Each edge is associated with a probability value that expresses the strength of the relationship. Determining the probability of an indirect association among a couple of entities is the same as computing the probability that a path exists between their nodes. The datasets are obtained from a network containing 11530 edges and 5220 nodes built around four genes responsible of Alzheimer's disease. Ten samples were extracted from the whole network each containing 50 graphs of increasing size (from 200 to 5000 nodes). For our test purposes we queried the probability that the genes HGNC_620 and HGNC_983 are related. Figure 1 presents the results of the experiments: the number of graphs for which the computation succeeded is reported on Figure 1(a), while Figure 1(b) reports the CPU time in seconds averaged over the graphs on which the algorithms succeeded as a function of the number of edges. The experimental results suggest that k-best does not improve with respect to exact because of the cost of keeping partial explanations sorted in sparse graphs, but Monte Carlo can solve twice as much problems than exact (up to 4000 edges). In terms of time, each algorithm performs almost like its ProbLog counterpart. With regard to the average absolute error, both k-best algorithms show a value of about 0.9%. Monte Carlo's average absolute error, however, is 4.9% for our implementation and 6.7% for ProbLog.

The artificial networks were used to evaluate the effective speedup in specific scenarios. The datasets contain graphs of increasing size that have different complexity with respect to the branching ratio and the length of paths between the terminal nodes. The graphs are built iteratively and are named after their shape: *lanes*, *branches* and *parachutes*. Lanes graphs, for example, gain a new parallel path a node longer than the previous graph. Branches are more complex because every step adds a new set of paths a node longer than before by forking at each node. Parachutes graphs are a trade-off between the two: they fork but each step introduces only one node (open paths fall back on existing nodes). Each dataset has a probability 0.3 on the edges and the path definition of lanes and parachutes contain 300 graphs, while branches only 25. Figure 2 shows an example for each dataset.



(a) Lanes. (b) Branches. (c) Parachutes.

Fig. 2. Examples of artificial graphs

Again, we queried the probability that a path exists between the terminal nodes (0 and 1) of the graphs. Figures 1(c), 1(d) and 1(e) show that in almost any case, our algorithms have performed better than their ProbLog equivalent, with Monte Carlo always being the fastest. The average absolute error for k-best and Monte Carlo is 0.001% and 3.170% respectively. ProbLog's Monte Carlo is not applicable because of the presence of a probability value in rules for *path*.

On the UWCSE dataset, Monte Carlo took 3.873 seconds to solve the problem with 20 students, while the algorithm CVE of [4] can solve at most the problem with 7 students and taking around 1000 seconds. For 7 students Monte Carlo takes 1.961 seconds and incurs in a 4.3% absolute error on the problem with 0 students, the only one for which we have the exact result (see Figure 1(f)). ProbLog's Monte Carlo was not applicable because the problem involves non ground probabilistic facts. ProbLog's *k*-best managed to solve the problem with 25 students thus resulting to be the fastest algorithm on this dataset. It incurs into an absolute error of 4.7% on the problem with 0 students.

The source code of the algorithms together with more details on the datasets and the experiments are available at the address `http://sites.google.com/a/unife.it/ml/acplint`.

Acknowledgements

This work has been partially supported by the Italian MIUR PRIN 2007 project No. 20077WWCR8.

References

1. De Raedt, L., Demoen, B., Fierens, D., Gutmann, B., Janssens, G., Kimmig, A., Landwehr, N., Mantadelis, T., Meert, W., Rocha, R., Santos Costa, V., Thon, I., Vennekens, J.: Towards digesting the alphabet-soup of statistical relational learning. In: NIPS*2008 Workshop on Probabilistic Programming (2008)
2. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discovery. In: 20th International Joint Conference on Artificial Intelligence, pp. 2468–2473. AAAI Press, Menlo Park (2007)
3. Kimmig, A., Santos Costa, V., Rocha, R., Demoen, B., De Raedt, L.: On the efficient execution of probLog programs. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 175–189. Springer, Heidelberg (2008)
4. Meert, W., Struyf, J., Blockeel, H.: CP-logic theory inference with contextual variable elimination and comparison to BDD based inference methods. In: De Raedt, L. (ed.) ILP 2009. LNCS, vol. 5989, pp. 96–109. Springer, Heidelberg (2010)
5. Riguzzi, F.: A top down interpreter for LPAD and CP-logic. In: Basili, R., Pazienza, M.T. (eds.) AI*IA 2007. LNCS (LNAI), vol. 4733, pp. 109–120. Springer, Heidelberg (2007)
6. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: Demoen, B., Lifschitz, V. (eds.) ICLP 2004. LNCS, vol. 3132, pp. 95–119. Springer, Heidelberg (2004)

Approximate Bayesian Computation for the Parameters of PRISM Programs

James Cussens

Department of Computer Science & York Centre for Complex Systems Analysis
University of York
Heslington, York, YO10 5DD, UK
jc@cs.york.ac.uk

Abstract. Probabilistic logic programming formalisms permit the definition of potentially very complex probability distributions. This complexity can often make learning hard, even when structure is fixed and learning reduces to parameter estimation. In this paper an approximate Bayesian computation (ABC) method is presented which computes approximations to the posterior distribution over PRISM parameters. The key to ABC approaches is that the likelihood function need not be computed, instead a 'distance' between the observed data and synthetic data generated by candidate parameter values is used to drive the learning. This makes ABC highly appropriate for PRISM programs which can have an intractable likelihood function, but from which synthetic data can be readily generated. The algorithm is experimentally shown to work well on an easy problem but further work is required to produce acceptable results on harder ones.

1 Introduction

In the Bayesian approach to parameter estimation a prior distribution for the parameters is combined with observed data to produce a posterior distribution. A key feature of the Bayesian approach is that the posterior provides a full picture of the information contained in prior and data: with an uninformative prior and little data we are not in a position to make confident estimates of the parameters and this will be reflected in a flat posterior. In contrast when much data is available the posterior will concentrate probability mass in small regions of the parameter space reflecting greater confidence in parameter estimates.

Despite its attractive features the Bayesian approach is problematic because in many cases computing or even representing the posterior distribution is very difficult. One area in which this is often the case is *statistical relational learning* (SRL). SRL formalisms combine probabilistic models with rich representation languages (often logical) which allows highly complex probabilistic models to be described. In particular, the *likelihood function* (the probability of observed data as a function of the model's parameters) is often intractable. This makes Bayesian and non-Bayesian parameter estimation difficult since the likelihood function plays a key role in both.

In this paper an *approximate Bayesian computation (ABC)* method is presented which approximates the posterior distribution over the parameters for a PRISM program with a given structure. The key feature of ABC approaches is that *the likelihood function is never calculated.* Instead synthetic datasets are generated and compared with the actually observed data. If a candidate parameter set generates synthetic datasets which are mostly 'close' to the real data then it will tend to end up with high posterior probability.

The rest of the paper is set out as follows. In Section 2 an account of approximate Bayesian computation is given. In Section 3 the essentials of PRISM programs are explained. Section 4 is the core of the paper where it is shown how to apply ABC to PRISM. Section 5 reports on initial experimental results and the paper concludes with Section 6 which includes pointers to future work.

2 Approximate Bayesian Computation

The ABC method applied in this paper is the ABC sequential Monte Carlo (ABC SMC) algorithm devised by Toni *et al* [1] and so the basic ideas of ABC will be explained using the notation of that paper. ABC approaches are motivated when the likelihood function is intractable but it is straightforward to sample synthetic data using any given candidate parameter set θ^*.

The simplest ABC algorithm is a rejection sampling approach described by Marjoram *et al* [2]. Since this is a Bayesian approach there must be a user-defined prior distribution $\pi(\theta)$ over the model parameters. Let x_0 be the observed data. If it is possible to readily sample from $\pi(\theta)$ then it is possible to sample from the *posterior distribution* $\pi(\theta|x_0)$ as follows: (1) sample θ^* from π, (2) sample synthetic data x^* from the model with its parameters set to θ^*, (i.e. sample from $f(x|\theta^*)$ where f is the likelihood function), (3) if $x_0 = x^*$ accept θ^*. The problem, of course, with this algorithm is that in most real situations the probability of sampling synthetic data which is exactly equal to the observed data will be tiny.

A somewhat more realistic option is to define a *distance function* $d(x_0, x^*)$ which measure 'how close' synthetic data x^* is to the real data x_0. x^* is now accepted at stage (3) above when $d(x_0, x^*) \leq \epsilon$ for some user-defined ϵ. With this adaptation the rejection sampling approach will produce samples from

$$\pi(\theta|d(x_0, x^*) \leq \epsilon)$$

As long as ϵ is reasonably small, this will be a good approximation to $\pi(\theta|x_0)$.

Choosing a value for ϵ is crucial: too big and the approximation to the posterior will be poor, too small and very few synthetic datasets will be accepted. A way out of this conundrum is to choose not one value for ϵ, but a *sequence* of decreasing values: $\epsilon_1, \ldots \epsilon_T$ $(\epsilon_1 > \cdots > \epsilon_T)$. This is the key idea behind the ABC sequential Monte Carlo (ABC SMC) algorithm:

> In ABC SMC, a number of sampled parameter values (called particles) $\{\theta^{(1)} \ldots \theta^{(N)}\}$, sampled from the prior distribution $\pi(\theta)$, is propagated through a sequence of intermediate distributions $\pi(\theta|d(x_0, x^*) \leq$

ϵ_t), $t = 1, \dots T-1$, until it represents a sample from the target distribution $\pi(\theta | d(x_0, x^*) \leq \epsilon_T)$. [1]

An important problem is how to move from sampling from $\pi(\theta | d(x_0, x^*) \leq \epsilon_t)$ to sampling from $\pi(\theta | d(x_0, x^*) \leq \epsilon_{t+1})$. In ABC SMC this is addressed via importance sampling. Samples are, in fact, not sampled from $\pi(\theta | d(x_0, x^*) \leq \epsilon_t)$ but from a *different sequence* of distributions $\eta_t(\theta)$. Each such sample θ_t is then weighted as follows $w_t(\theta_t) = \frac{\pi(\theta_t | d(x_0, x^*) \leq \epsilon_t)}{\eta_t(\theta_t)}$. η_1, the first distribution sampled from, is chosen to be the prior π. Subsequent distributions η_t are generated via a user-defined perturbation kernels $K_t(\theta_{t-1}, \theta_t)$ which perform moves around the parameter space.

These are the basic ideas of the ABC SMC algorithm; full details are supplied by Toni *et al* [1] (particularly Appendix A). As a convenience the description of the ABC SMC algorithm supplied in that paper is reproduced (almost verbatim) in Fig. 1.

S1 Initialise $\epsilon_1, \dots \epsilon_T$.
Set the population indicator $t = 0$.

S2.0 Set the particle indicator $i = 1$.

S2.1 If $t = 0$, sample θ^{**} independently from $\pi(\theta)$.
If $t > 0$, sample θ^* from the previous population $\{\theta_{t-1}^{(i)}\}$ with weights w_{t-1} and perturb the particle to obtain $\theta^{**} \sim K_t(\theta | \theta^*)$, where K_t is a perturbation kernel.
If $\pi(\theta^{**}) = 0$, return to **S2.1**
Simulate a candidate dataset $x^*_{(b)} \sim f(x|\theta^{**})$ B_t times ($b = 1, \dots, B_t$) and calculate $b_t(\theta^{**}) = \sum_{b=1}^{B_t} 1(d(x_0, x^*_{(b)}) \leq \epsilon_t)$.
If $b_t(\theta^{**}) = 0$, return to **S2.1**.

S2.2 Set $\theta_t^{(i)} = \theta^{**}$ and calculate the weight for particle $\theta_t^{(i)}$,

$$w_t^{(i)} = \begin{cases} b_t(\theta_t^{(i)}), & \text{if } t = 0 \\ \frac{\pi(\theta_t^{(i)}) b_t(\theta_t^{(i)})}{\sum_{j=1}^{N} w_{t-1}^{(j)} K_t(\theta_{t-1}^{(j)}, \theta_t^{(j)})} & \text{if } t > 0 \end{cases}$$

If $i < N$ set $i = i + 1$, go to **S2.1**

S.3 Normalize the weights.
If $t < T$, set $t = t + 1$, go to **S2.0**

Fig. 1. ABC SMC algorithm reproduced from Toni *et al* [1]

Note, from Fig. 1, that rather than generate a single dataset from $f(x|\theta^{**})$, B_t datasets are sampled where B_t is set by the user. The quantity $b_t(\theta^{**})$ is the count of synthetic datasets which are within ϵ_t. The intuitive idea behind ABC SMC is that a particle θ^{**} that generates many synthetic datasets 'close' to x_0, will get high weight and is thus more likely to be sampled for use at the next iteration.

3 PRISM

PRISM (PRogramming In Statistical Modelling) [3] is a well-known SRL formalism which defines probability distributions over possible worlds (Herbrand models). The probabilistic element of a PRISM program is supplied using *switches*. A switch is syntactically defined using declarations such as those given in Fig. 2 for switches `init`, `tr(s0)`, `tr(s1)`, `out(s0)` and `out(s1)`.

The declaration in Fig. 2 for `init`, for example, defines an infinite collection of independent and identically distributed binary random variables $\mathtt{init}_1, \mathtt{init}_2, \ldots$ with values `s0` and `s1` and with distribution $P(\mathtt{init}_i = \mathtt{s0}) = 0.9$, $P(\mathtt{init}_i = \mathtt{s1}) = 0.1$ for all i.

The rest of a PRISM program is essentially a Prolog program. Switches provide the probabilistic element via the built-in predicate `msw/2`. Each time a goal such as `:- msw(init,S)` is called the variable `S` is instantiated to `s0` with probability 0.9 and `s1` with probability 0.1. If this were the ith call to this goal then this amounts to sampling from the variable $\mathtt{init}_i$.

Although later versions of PRISM do not require this, it is convenient to specify a *target predicate* where queries to the target predicate will lead to calls to `msw/2` goals (usually via intermediate predicates). The target predicate is thus a *probabilistic predicate*: the PRISM program defines a distribution over instantiations of the variables in target predicate goals. For example, in the PRISM program in Fig. 2 which implements a hidden Markov model, `hmm/1` would be the target predicate. A query such as `:- hmm(X).` will lead to instantiations such as `X = [a,a,b,a,a]`.

```
values(init,[s0,s1]).       % state initialization
values(out(_),[a,b]).       % symbol emission
values(tr(_),[s0,s1]).      % state transition

hmm(L):-                        % To observe a string L:
   msw(init,S),                 %   Choose an initial state randomly
   hmm(1,5,S,L).                %   Start stochastic transition (loop)

hmm(T,N,_,[]):- T>N,!.          % Stop the loop
hmm(T,N,S,[Ob|Y]) :-            % Loop: current state is S, current time is T
   msw(out(S),Ob),              %   Output Ob at the state S
   msw(tr(S),Next),             %   Transit from S to Next.
   T1 is T+1,                   %   Count up time
   hmm(T1,N,Next,Y).            %   Go next (recursion)

:- set_sw(init,   [0.9,0.1]), set_sw(tr(s0), [0.2,0.8]),
   set_sw(tr(s1), [0.8,0.2]), set_sw(out(s0),[0.5,0.5]),
   set_sw(out(s1),[0.6,0.4]).
```

Fig. 2. PRISM encoding of a simple 2-state hidden Markov model (this example is distributed with the PRISM system)

The switch probabilities are the parameters of a PRISM program and the data used for parameter estimation in PRISM will be a collection of ground instances of the target predicates which are imagined to have been sampled from the unknown 'true' PRISM program with the 'true' parameters. PRISM contains a built-in EM algorithm for maximum likelihood parameter and maximum a posteriori (MAP) estimation [4]. In both cases a point estimate for each parameter is provided. In contrast here an approximate sample from the posterior distribution over parameters is provided by a population of particles.

4 ABC for PRISM

To apply the ABC SMC algorithm it is necessary to choose: (1) a prior distribution for the parameters, (2) a distance function, (3) a perturbation kernel and (4) also the specific experimental parameters such as the sequence of ϵ_t, etc. The first of these three are dealt with in the following three sections (4.1–4.3). The choice of experimental parameters is addressed in Section 5.

4.1 Choice of Prior Distribution

The 'obvious' prior distribution is chosen. Each switch has a user-defined Dirichlet prior distribution and the full joint prior distribution is just a product of these. This is the same as the prior used for MAP estimation in PRISM [4, §4.7.2]. To sample from this prior it is enough to sample from each Dirichlet independently. Sampling from each Dirichlet is achieved by exploiting the relationship between Dirichlet and Gamma distributions. To produce a sample $(p_1, \ldots, p_k)$ from a Dirichlet with parameters $(\alpha_1, \ldots \alpha_k)$, values z_i are sampled from Gamma$(\alpha_i, 1)$ and then p_i is set to $z_i/(z_1 + \cdots + z_k)$. The z_i are sampled using the algorithm of Cheng and Feast [5] for $\alpha_i > 1$ and the algorithm of Ahrens and Dieter [6] for $\alpha_i \leq 1$. Both these algorithms are given in [7]. For Dirichlet distributions containing small values of α_i, numerical problems sometimes produced samples where $p_i = 0$ for some i which is wrong since the Dirichlet has density 0 for any probability distribution containing a zero value. This problem was solved by the simple expedient of not choosing small values for the α_i!

4.2 Choice of Distance Function

The basic ABC approach leads to a sample drawn from $\pi(\theta|d(x_0, x^*) \leq \epsilon)$ rather than $\pi(\theta|x_0)$. For this to be a good approximation it is enough that $f(x^*|\theta) \approx f(x_0|\theta)$ for all x^* where $d(x_0, x^*) \leq \epsilon$. With this in mind d is defined as follows. Let $P(x_0)$ be the empirical distribution defined by the real data x_0. $P(x_0)$ assigns a probability to every possible ground instance of the target predicate. This probability is just the frequency of the ground instance in the data divided by the total number of datapoints. $P(x^*)$ is the corresponding empirical distribution for fake data x^*. Both $P(x_0)$ and $P(x^*)$ can be viewed as (possibly countably infinite) vectors of real numbers. The distance between x and x^* is then defined to be the squared Euclidean distance between $P(x_0)$ and $P(x^*)$. Formally:

$$d(x, x^*) = \sum_{i \in I} (P(x_0)(i) - P(x^*)(i))^2 \tag{1}$$

where I is just some (possibly countably infinite) index set for the set of all ground instances of the target predicate. In practice most terms in the sum on the RHS of (1) will be zero, since typically most ground instances appear neither in the real data nor in fake datasets.

4.3 Choice of Perturbation Kernel

Recall that each particle θ^* defines a multinomial distribution of the appropriate dimension for each switch of the PRISM program. The perturbation kernel $K_t(\theta|\theta^*)$ has two stages. Firstly, Dirichlet distributions are derived from θ^* by multiplying each probability in θ^* by a global value α_t where $\alpha_t > 0$. Secondly, a new particle is sampled from this product of Dirichlets using exactly the same procedure as was used for sampling from the original prior distribution. Large values of α_t will make small moves in the parameter space likely (since the Dirichlet distributions will be concentrated around θ^*) and small values of α_t will encourage larger moves. An attractive option is to start with small values of α_t to encourage exploration of parameter space and to progressively increase α_t in the hope of convergence to a stable set of particles giving a good approximation to the posterior.

5 Experimental Results

The ABC SMC algorithm has been implemented as a PRISM program which is supplied in the supplementary materials. PRISM 2.0 beta 4, kindly supplied by the PRISM developers, was used. As an initial test, ABC was done for the simplest possible parameter estimation problem. A PRISM program representing a biassed coin ($P(\text{heads}) = 0.7$, $P(\text{tails}) = 0.3$) was written and data of 100 simulated tosses were produced. This resulted in 67 heads and 33 tails. ABC was run several times with the following (more or less arbitrarily chosen) parameters: prior distribution $\pi(\theta) = \text{Dir}(1, 1)$, sequence of thresholds $\epsilon = (0)$, number of synthetic datasets $B_t = 50$, perturbation kernel parameter $\alpha_t = 2$, number of particles $T = 50$ and population size $N = 50$. As expected the final population of (weighted) particles were always concentrated around the maximum likelihood estimate $P(\text{heads}) = 0.67$, $P(\text{tails}) = 0.33$. Here are the 4 most heavily weighted particles with their weights from one particular run: $(0.646, 0.3534), (w = 0.051)$, $(0.647, 0.353), (w = 0.044)$, $(0.667, 0.332), (w = 0.044)$, $(0.62, 0.38), (w = 0.037)$. Estimates of the posterior mean were similar for different ABC runs: here are such estimates from 5 runs: $(0.663, 0.337)$,

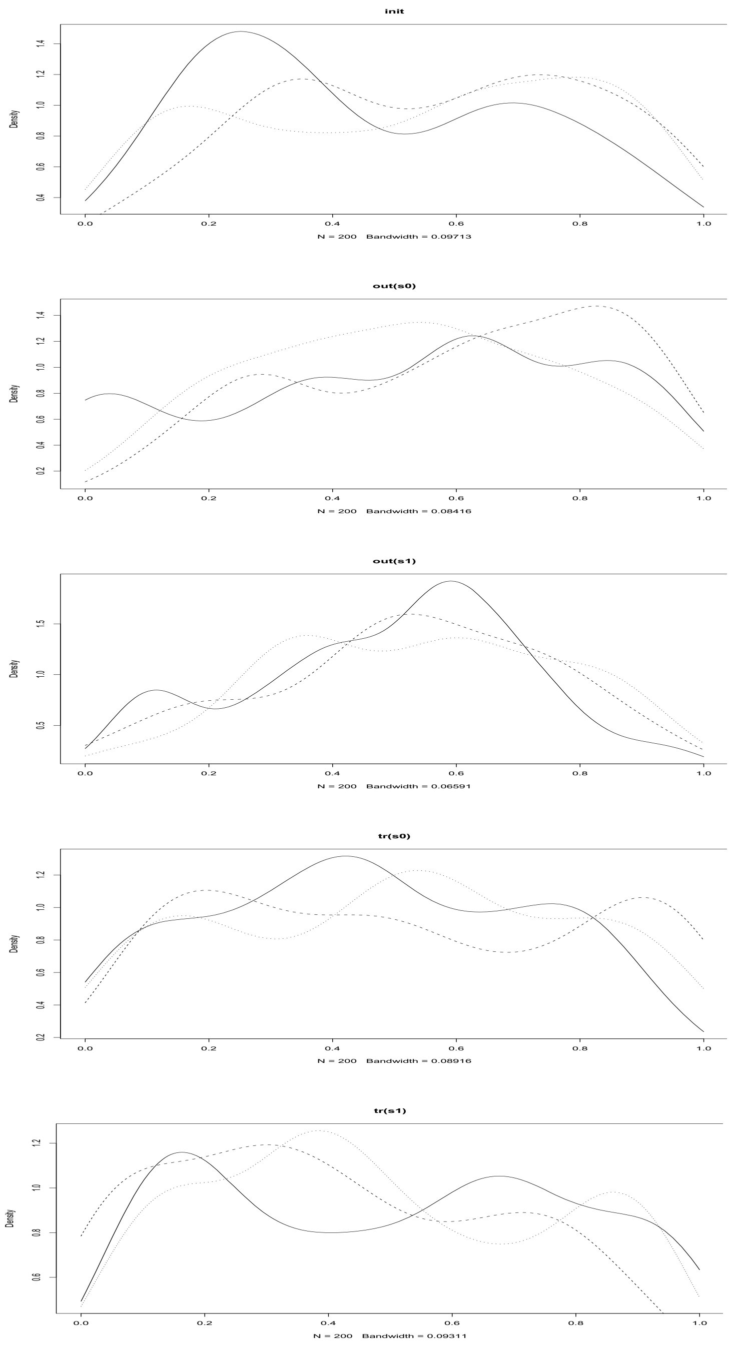


Fig. 3. Posterior distributions for HMM switch probabilities `init`, `out(s0)`, `out(s1)`, `tr(s0)`, `tr(s1)` as estimated by three different runs of ABC.

$(0.655, 0.345)$, $(0.664, 0.336)$, $(0.667, 0.333)$, $(0.677, 0.323)$. Note that, in this trivial problem, successful parameter estimation was possible by going directly for a zero distance threshold $\epsilon = (0)$.

For a more substantial test, 100 ground `hmm/1` atoms were sampled from the PRISM encoded HMM show in Fig. 2. $\text{Dir}(1, 1)$ priors were used for all 5 switches. The experimental parameters were $\alpha_t = 10$, $B_t = 100$, $N = 200$ and $\epsilon = (0.1, 0.05)$. Ideally, different runs of ABC should generate similar approximations to posterior quantities. To look into this, marginal posterior distributions for the probability of the first value of each of the 5 switches were estimated using 3 different ABC runs. The results are shown in Fig. 3. These plots were produced using the `density` function in R with the final weighted population of particles as input. There is evident variation between the results of the 3 runs, but similarities also. All 3 densities for `init` contain two local modes, all 3 for `out(s1)` put most mass in the middle, all 3 for `tr(s0)` have a fairly even spread apart from extreme values.

6 Conclusions and Future Work

This paper has described ABC SMC for PRISM programs and has shown some initial results for a working implementation. Evidently, considerably more experimentation and theoretical analysis is required to provide reliable approximations to posterior quantities using ABC SMC. In the experiments reported above the perturbation kernel K_t did not vary with t. It is likely that better results are possible by reducing the probability of big perturbations as t increases. In addition the choice for the sequence ϵ_t thresholds was fairly arbitrary. Finally, it may be that better results are achievable by throwing more computational resources at the problem: most obviously increasing the number of particles, but also by lengthening the sequence of ϵ_t thresholds to effect a smoother convergence to the posterior.

Another avenue for improvement is the choice of distance function. The function introduced in Section 4.2 is a generic function that is applicable to any PRISM program. It seems likely that domain knowledge could be used to choose domain-specific distance functions which reflect the 'real' difference between different ground atoms. The function used here treats all distinct pairs of ground atoms as equally different which will not be appropriate in many cases.

References

1. Toni, T., Welch, D., Strelkowa, N., Ipsen, A., Stumpf, M.P.: Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. Journal of the Royal Society Interface 6(31), 187–202 (2009)
2. Marjoram, P., Molitor, J., Plagnol, V., Tavaré, S.: Markov chain Monte Carlo without likelihoods. Proceedings of the National Academy of Science 100, 15324–15328 (2003)
3. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. Journal of Artificial Intelligence Research 15, 391–454 (2001)

4. Sato, T., Zhou, N.F., Kameya, Y., Izumi, Y.: PRISM User's Manual, Version 1.12.1 (2009)
5. Cheng, B., Feast, G.: Some simple gamma variate generators. Applied Statistics 28, 290–295 (1979)
6. Ahrens, J., Dieter, U.: Computer methods for sampling from gamma, beta, Poisson and binomial distributions. Computing 12, 223–246 (1974)
7. Robert, C.P., Casella, R.: Monte Carlo Statistical Methods, 2nd edn. Springer, New York (2004)

Probabilistic Rule Learning

Luc De Raedt and Ingo Thon

Department of Computer Science, Katholieke Universiteit Leuven, Belgium

Abstract. Traditionally, rule learners have learned deterministic rules from deterministic data, that is, the rules have been expressed as logical statements and also the examples and their classification have been purely logical. We upgrade rule learning to a probabilistic setting, in which both the examples themselves as well as their classification can be probabilistic. The setting is incorporated in the probabilistic rule learner ProbFOIL, which combines the principles of the relational rule learner FOIL with the probabilistic Prolog, ProbLog. We report also on some experiments that demonstrate the utility of the approach.

1 Introduction

Rule learners are amongst the most popular and easiest to use machine learning systems. They learn logical rules from deterministic examples but do not really take into account uncertainty. On the other hand, the graphical model and statistical relational learning community are able to reason about uncertainty but have not yet contributed many approaches to learning logical rules. This paper wants to alleviate this situation by introducing a novel probabilistic rule learning setting. In this setting, logical rules are learned from probabilistic data in the sense that both the examples themselves and their classifications can be probabilistic.

As a motivating example that we will use throughout this paper, consider the following windsurfing problem. It is inspired by Quinlan's play-tennis example. The difference between playing tennis and going windsurfing is that windsurfing typically needs to be planned ahead of time, say on the previous day. The effect is that the weather conditions at the next day will still be uncertain at the time of deciding whether to go surfing or not. The forecast might state that tomorrow the probability of precipitation (pop) is 20%, the wind will be strong enough with probability 70%, and the sun is expected to shine 60% of the time, which could be represented by the facts

```
0.2::pop(t). 0.7::windok(t). 0.6::sunshine(t).
```

where the `t` indicates the identifier for the example. Past experience in this case would consist of such descriptions together with a probability value for the target predicate (e.g., `0.7::surfing(t)`), which could indicate, for instance, the percentage of persons in our team that enjoyed the activity, the percentage of time that we enjoyed the surfing, etc. This type of data can be represented using a traditional attribute-value table, where the attributes are all Boolean and the values are the probabilities with which the attribute is true.

The probabilistic rule learning problem, introduced in this paper, is now to induce a set of rules that allows one to predict the probability of the example from its description. For instance, for the surfing example, the following rules could be induced:

```
surfing(X) :- not pop(X), windok(X).
surfing(X) :- not pop(X), sunshine(X).
```

where the argument `X` specifies the identifier of the example. The first rule states that if the expected precipitation is low and the wind is ok, the surfing is likely to be good. There is thus a declarative logical reading of these rules, but also a probabilistic one: the lower the precipitation is and the higher the probability of windok and sunshine the higher the probability that the surfing will be enjoyable. Using the description of the example (`0.2::pop(t). 0.7::windok(t). 0.6::sunshine(t).`), we can compute the probability of `surfing(t)` under this hypothesis. Assuming all facts in the description are independent, this reduces to

$$\begin{aligned} P(\texttt{surfing(t)}) &= P((\neg\texttt{pop(t)} \wedge \texttt{windok(t)}) \vee (\neg\texttt{pop(t)} \wedge \texttt{sunshine(t)}) \\ &= P((\neg\texttt{pop(t)} \wedge \texttt{windok(t)}) \\ &\qquad \vee(\neg\texttt{pop(t)} \wedge \texttt{sunshine(t)} \wedge \neg\texttt{windok(t)})) \\ &= 0.8 \times 0.7 + 0.8 \times 0.6 \times 0.3 = 0.704 \end{aligned}$$

where the rewriting is needed to make the two rules mutually exclusive.

Observe that although the windsurfing example is a toy example, this type of probabilistic data arises naturally in many application domains, such as robotics, vision, natural language processing and the life sciences. For instance, in a vision context there might be uncertainty about the identity, features or class of the object just observed; cf. also the experimental section. In the literature this kind of evidence is known as *soft evidence*, which has been studied in the context of probabilistic graphical models. The goal is there typically to calculate the most likely state for the remaining distributions [1].

One might want to tackle the probabilistic rule learning problem with traditional rule learning techniques. Two approaches come to mind, but turn out to be problematic. First, one might simply ignore the probabilities in the example and take the most likely value instead. Applied to the above example, this would yield the positive example (because $0.7 \geq 0.5$) with the description (`not pop(t). windok(t). sunshine(t).`). Even though the two rules – in this case – would predict the correct class, this approach necessarily leads to a loss of information as only 0/1 values can be predicted, which also results in a loss of prediction accuracy. Secondly, one might want to turn the probabilistic example into a set of deterministic ones by sampling instances from the example. Indeed, for each fact and the target predicate in the example one could sample possible deterministic facts. Each fact $p :: f$ would be sampled as true with probability p

and as false with probability $1 - p$. This causes two problems. First, the generated examples may be inconsistent with one another as generated examples may be identical except for their target class. This will cause problems to many rule learners. Second, it requires one to sample a lot of deterministic instances from a probabilistic one to avoid overfitting. If there are n facts and approximating one attribute up to a certain accuracy requires k samples, approximating the entire example with the same accuracy will require k^n deterministic examples. Although from these examples one could in principle learn the rules, and one could use the rules to compute the probability of an example by classifying a set of deterministic samples from that example, it should be clear that this approach will lead to combinatorial problems.

A further approach that one might try is to consider the learning task as a regression problem. In the inductive logic programming community several techniques have been developed that integrate rule learning with regression, cf. [2,3]. These can also be applied to learn a set of Prolog rules that would compute the probability of an example. However, these rules typically contain a (complicated) equation that allows one to compute the probability or value of the example in terms of the probability of the features. While the performance of such approaches is typically good, the resulting rules are much harder to interpret than the logical ones that we induce. Furthermore, this type of approach is not really integrated in a probabilistic logical or statistical relational learning system.

This paper is organized as follows: In Section 2, we review ProbLog and formally introduce the problem; in Section 3, we analyze the problem of probabilistic rule learning and compare it to the deterministic case; in Section 4, we introduce the probabilistic rule learner ProbFOIL which integrates principles of FOIL and ProbLog; in Section 5, we report on some preliminary experiments, and finally, in Section 6, we conclude.

2 Problem Specification

We first introduce ProbLog, a probabilistic Prolog [4,5]. A ProbLog program consists of a set of definite clauses D and a set of probabilistic facts $p_i :: c_i$, which are facts c_i labeled with the probability p_i that their ground instances $c_i\theta$ are true. It is also assumed that the probabilities of all ground instances $c_i\theta$ are mutually independent.

Given a finite set of possible substitutions $\{\theta_{j1}, \ldots \theta_{ji_j}\}$ for each probabilistic fact $p_j :: c_j$, a ProbLog program $T = \{p_1 :: c_1, \cdots, p_n :: c_n\} \cup D$ defines a probability distribution

$$P(L \mid T) = \prod\nolimits_{c_i\theta_j \in L} p_i \prod\nolimits_{c_i\theta_j \in L_T \setminus L} (1 - p_i)$$

over ground subprograms $L \subseteq L_T = \{c_1\theta_{11}, \ldots c_1\theta_{1i_1}, \cdots, c_n\theta_{n1}, \ldots, c_n\theta_{ni_n}\}$. ProbLog is then used to compute the *success probability*

$$P_s(T \models q) = \sum\nolimits_{L \subseteq L_T} P(q|L \cup D) \cdot P(L|T)$$

of a query q in a ProbLog program T, where $P(q|L \cup D) = 1$ if there exists a θ such that $L \cup D \models q\theta$, and $P(q|L \cup D) = 0$ otherwise. In other words, the success probability of query q corresponds to the probability that the query q is *entailed* using the background knowledge together with a randomly sampled set of ground probabilistic facts. An example ProbLog program and query is shown above in the `surfing` example. For more details on ProbLog as well as on its efficient implementation, we refer to [5].

We are now able to formalize the problem of *inductive probabilistic logic programming* or *probabilistic rule learning* as follows:

Given:

1. $E = \{(x_i, p_i) | x_i$ a ground fact for the unknown target predicate t; $p_i \in [0,1]$ the target probability of $x_i\}$, the set of examples;
2. a background theory B containing information about the examples in the form of a probabilistic ProbLog program;
3. a loss function $loss(H, B, E)$, measuring the loss of a hypothesis H (that is, a set of clauses) w.r.t. B and E;

Find: $\arg\min_H loss(H, B, E) = \arg\min_H \sum_{e_i \in E} |P_s(B \cup H \models e) - p_i|$

This loss function aims at minimizing the absolute difference between the predictions and the observations. The reason for this choice is, on the one hand, that it is the simplest possible choice and, on the other hand, that well-known concepts and notions from rule learning and classification carry over to the probabilistic case when this loss function is used as we shall show.

There are several interesting observations about this problem setting. First, it generalizes both traditional rule learning and inductive logic programming to a probabilistic setting. The propositional case illustrated in the windsurfing example is an example of probabilistic rule learning. Furthermore, when the background theory contains also relations and possibly clauses defining further predicates we obtain an inductive probabilistic logic programming setting. In both cases, the original setting is obtained by assuming that the background theory is purely logical and having as only values for the examples 1 and 0; 1 corresponding to the positive examples and 0 to the negative ones. This is in line with the theory of probabilistic logic learning [6] and the inductive logic programming setting obtained would be that of learning from entailment because examples are facts that are probabilistically entailed by the theory.

Second, as in traditional symbolic learning the goal is to find a set of logical rules that satisfy certain constraints, while the rules themselves do not possess any parameters. To the best of the authors' knowledge this problem has not been studied before. It is also interesting to position this problem in the context of the literature on uncertainty in artificial intelligence. There one typically makes a distinction between parameter learning and structure learning, the latter being an extension of the former in that also in structure learning the parameters have to be estimated. The probabilistic rule learning problem introduced above is in a sense dual to the parameter estimation problem. Indeed, when estimating parameters, the structure of the model is assumed to be given and fixed, while here the parameters (the probability values) are fixed and the structure, that is, the rules are to be learned.

It is of course also possible to extend the problem setting so that induced rules may contain new predicates defined by probabilistic facts with unknown probability values. This can be realized by adding to each clause h :- $b_1, ..., b_m$ in a hypothesis a new literal l where l would be defined through a probabilistic predicate that would be unique to the clause. This type of extension would require both rule learning and parameter estimation. Although we are currently exploring this setting, we will – in the present paper – not consider this setting any further and focus instead on the pure probabilistic rule learning problem because it is this setting that directly upgrades the well established rule-learning problem.

3 Analysis

A key difference between the probabilistic and the deterministic setting is that each example e_i now has a target probability p_i as opposed to a 1/0 value. Furthermore, while in the deterministic case one obtains a 1/0 error, the probabilistic case is more subtle. To clarify this, we use p_i to denote the positive and $n_i = 1 - p_i$ the negative part of the example e_i, while $p_{h,i}$ and $n_{h,i} = 1 - p_{h,i}$ denote the positive and negative prediction w.r.t. the hypothesis h, and introduce the following quantities:

1. the true positive part $tp_i = min(p_i, p_{h,i})$,
2. the true negative part $tn_i = min(n_i, n_{h,i})$,
3. the false positive part $fp_i = max(0, n_i - tn_i)$, and
4. the false negative part $fn_i = max(0, p_i - tp_i)$.

These notions are graphically illustrated in Figure 1. If the prediction is perfect, that is, if $p_{h,i} = p_i$, then $n_{h,i} = n_i$, then the true positive and negative parts are maximal and the false positive and negative part are minimal, that is, 0. However, if $p_{h,i} > p_i$ the hypothesis h overestimates the positive part of the example, and hence, the true positive part is still p_i but the false positive part will be non-zero. Dually, if $p_{h,i} < p_i$, the true negative part is still n_i but the false negative part will be non-zero. Furthermore, let us denote by

$$TP = \sum_i tp_i; \quad TN = \sum_i tn_i; \quad FP = \sum_i fp_i; \quad FN = \sum_i fn_i$$

that is, the sum of the tp_i, tn_i, fp_i and fn_i, where the sum is taken over all examples in the dataset and by

$$M = |E|; \quad P = \sum_i p_i; \quad N = \sum_i n_i$$

These notions can be displayed in a contigency table, cf. Table 1. It should be clear that this probabilistic contingency table and the above introduces notions

Table 1. A probabilistic contingency table

	Predicted True	Predictive False	
Real True	TP	FN	P
Real False	FP	TN	N
			M

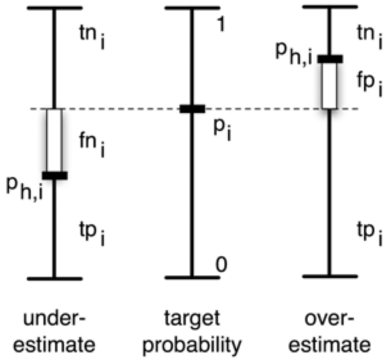


Fig. 1. The true and false positive and negative part of a single example

directly generalize the deterministic case. To see this, consider that any positive example classified as such will contribute a value of $tp_i = 1$ to TP and $fn_i = 0$ to FN, and any positive example classified as negative will contribute $tp_i = 0$ to TP and $fn_i = 1$ to FN. Thus we have the following property.

Property 1. The probabilistic contingency table generalizes the deterministic one.

The different notions are graphically displayed in Figure 2, in which the x-axis contains the examples and the y-axis their probability and all the examples are ordered according to increasing target probability. The areas then denote the respective rates. The deterministic case is illustrated in the figure 2 (right), which shows that in this case the examples take on 1/0 values. Because the TP and FP rates form the basis for ROC analysis, traditional ROC analysis, as used in rule learning and classification systems can be applied to the probabilistic rule learning setting that we study in this paper. The reason is that any given hypothesis corresponds to a point in ROC space and can be interpreted in a similar way as in traditional rule learning. Therefore, ROC analysis techniques and measures such as AUC essentially carry over to the probabilistic case.

Using these notions we also define *precision*, *recall* (true positive rate) and *accuracy* using the standard formulas.

$$precision = \frac{TP}{TP+FP} \qquad m\text{-}estimate = \frac{TP+m\cdot\frac{P}{N}}{TP+FP}$$

$$recall = \frac{TP}{TP+FN} \qquad accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

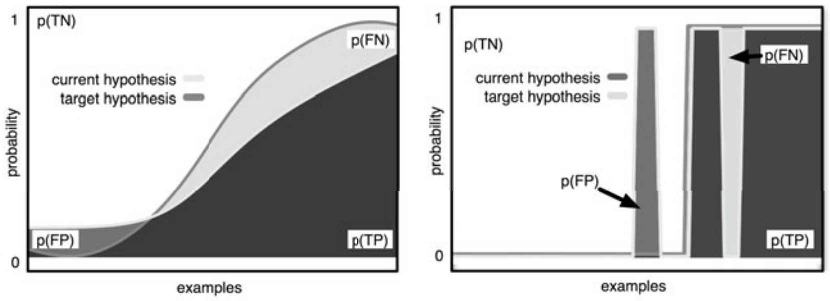


Fig. 2. The true and false positive part of an entire dataset for the probabilistic (left) case, and for the deterministic case (right)

4 ProbFOIL: A Probabilistic First Order Rule Learner

We now develop a probabilistic rule learner called ProbFOIL, which is able to induce probabilistic logic programs from examples. The rule learner is simple in that it follows the standard and generally accepted principles of rule learners (as described by [7,8]) but does not yet incorporate advanced pruning strategies. Instead of incorporating very elaborated pruning techniques we restrict ourselves to incremental reduced error pruning which is known to have a good trade-off between complexity and quality and the m-estimate which is known to give reliable results in the presence of noise.

While developing ProbFOIL (Algorithm 1) we started from the generic separate and conquer paradigm (sometimes called the sequential covering algorithm) and modified it as little as possible. Essentially, the algorithm repeatedly adds clauses to the hypothesis in the outer loop until adding further clauses decreases the quality of the hypothesis. Furthermore, while searching ffor the next clause (lines 5-8) it searches greedily according to some local scoring function in which it repeatedly adds literals to the current clause until some local stopping criterion is satisfied. To determine the possible literals, a refinement operator ρ is applied to the current clause; cf. [6]. We also employ list notation for the bodies of the rules, where the notation $[b, l]$ denotes the result of appending the literal l to the body b. The post-processing step of the rule in lines 9-11 implements a kind of post-pruning akin to that in IREP [9]. The resulting algorithm is very much like the standard rule-learning algorithm known from the literature; cf. [7,8].

While the algorithm is similar to that of typical rule-learners, it is important to realize that there are also some subtleties. First, adding clauses to the hypothesis for the target predicate is a monotonic operation, that is, it can only increase the probability of an individual example because adding a clause results in extra possibilities for proving that the example is true. More formally:

Property 2. For all hypotheses H_1, H_2: $H_1 \subseteq H_2 \rightarrow TP(H_1) + FP(H_1) \leq TP(H_2) + FP(H_2)$.

Interpreted using Figure 2, adding clauses to a hypothesis can only increase the FP and TP regions, that is, move them upwards, and at the same time reduce the FN and TP ones. This explains why ProbFOIL stops adding clauses to the hypothesis when adding the rule found last does not result in a better global score. This is akin to the standard stopping criterion employed in many rule learners. As the global scoring function we employ *accuracy*(H).

Secondly, notice that specializing a clause, that is, adding literals to a clause can only decrease the probability of examples (and hence, decrease the TP and FP regions).

Property 3. For all hypotheses H and clauses $h \leftarrow l_1, ..., l_n$ and literals l: $TP(H \cup \{h \leftarrow l_1, ..., l_n\}) + FP(H \cup \{h \leftarrow l_1, ..., l_n\}) \leq TP(H \cup \{h \leftarrow l_1, ..., l_n, l\}) + FP(H \cup \{h \leftarrow l_1, ..., l_n, l\})$

Thirdly, while traditional deterministic rule learners typically manipulate also the set of examples (e.g. deleting the already covered examples), our probabilistic rule learner takes into account all examples all of the time. In the deterministic case, deleting the already covered examples is warranted because if one rule in the hypothesis covers the example, the overall hypothesis will cover the example. In the probabilistic case, this is more subtle as a given rule may only cover part of the example, and therefore a multi-rule hypothesis may be needed to cover the full positive part of an example. Our algorithm takes this into account in the heuristics used in its inner loop, where it will make decisions based on the extra parts of the examples that become covered by the new rule. In terms of the visualization in Figure 2, this is the difference between the old and new TP and FP parts. However, because each rule may only cover fractions of the examples, we use the difference in m-estimate, i.e.,

$$\text{localscore}(H, c) = \text{m-estimate}(H \cup \{c\}) - \text{m-estimate(H)}$$

The m-estimate, on which the local scoring function is based, is a variant of precision that is more robust against noise in the training data.

Finally, ProbFOIL stops refining rules when the current rule does not cover any extra negative part any more, or when it does not cover any extra positive part any more. More formally,

$$\text{localstop}(H, c) = (TP(H \cup \{c\}) - TP(H) = 0) \vee (FP(\{c\}) = 0)$$

It should also be clear that standard extensions of rule-learning, such as those for dealing with multiple classes, using beam-search, and look-ahead, can easily be incorporated in ProbFOIL.

5 Experiments

We demonstrate our approach in two experiments. In the first experiment we learned the rules for the `surfing` example, in the second experiment we learned the underlying rules in the *Eulisis* game starting from image data. We used the YAP-ProbLog implementation and computed all scores using exact inference. All experiments were performed on a 3GHz Machine with 2GB of Ram.

Algorithm 1. The ProbFOIL algorithm

```
1: $h := t(X_1, \dots, X_n)$ where $t$ is the target predicate and the $X_i$ distinct variables;
2: $H := \emptyset$; $b := []$; $c := (h \leftarrow b)$;
3: **repeat**
4:     $b := []$; initially the body of the rule is empty;
5:     **while** $\neg$localstop($H$,$h \leftarrow b$) **do**                    ▷ Grow rule
6:         $l := \arg\max_{l \in \rho(h \leftarrow b)}$ localscore($h \leftarrow [b, l]$)
7:         $b := [b, l]$
8:     let $b = [l_1, \dots, l_n]$
9:     $i := \arg\max_i$ localscore($H, h \leftarrow l_1, ..., l_i$);
10:    $c := p(X_1, \dots, X_n) \leftarrow l_1, \dots, l_i$;
11:    **if** globalscore($H$) $<$ globalscore($H \cup \{c\}$) **then**
12:        $H := H \cup \{c\}$
13: **until** globalscore($H$) $>$ globalscore($H \cup \{c\}$)
14: **return** $H$
```

5.1 Surfing

For the surfing example we generated a dataset of 20 training examples starting from the two clauses listed in the introduction. The probabilities of the features were randomly initialized, and we were able to rediscover the original rule set. The runtime was less than 40 seconds. Afterwards we tried to rediscover the original rule-set. The rule-set inferred was

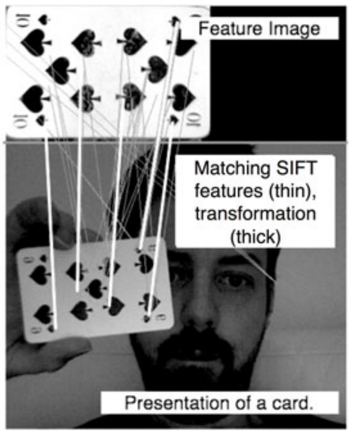
```
1.) surfing(I):- windok(I),\+ pop(I)
2.) surfing(I):- \+ pop(I),sunshine(I).
3.) % surfing(I):- sunshine(I),\+ windok(I).
```

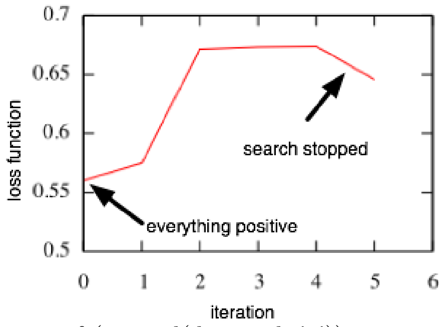
Please note that when calculating the score of a rule `I` is ground. This allows to calculate the probability of negated literals. The last rule decreases the total accuracy therefor it is in accordance to the algorithm not added to the final rule-set and search is stopped. Runtime was less then 40 seconds.

5.2 Eulisis

For the second experiment we used the game of *Eulisis* [10]. *Eulisis* is a game where the dealer has a secret set of rules in mind. For each partial sequence of cards, the rules specify which cards are valid extensions of the sequence and which ones are not. After a card is played, the players are told whether the card is a valid extension or not. Using this information they have to guess the set of secret rules. The concept represented by the rules has to be expressed in terms of the *suit*, *rank*, *color* of the card, and whether it is is *even* or a *face* card.

We played this game against the computer by taking actual images of the played cards (cf. also Figure 2). First, we presented the computer a sequence of 32 cards in a random order. For each card in the sequence the computer is told whether the card is a positive extension of the present partial sequence or not, and hence, each card played together with the previous card yields one example.





0.13::sift(26,card(diamonds,'9')).
0.24::sift(26,card(heart,'9')).
0.09::sift(26,card(spades,'10')).
0.40::sift(26,card(spades,'9')).

Fig. 3. *SIFT* feature matches for spades nine and spades 10 (left). All matches for the 26th card which is a spades 9 bottom (right,bottom). Accuracy of the rule-set after each iteration on the red implies even dataset (right,top).

The cards are classified using *SIFT* (sale-invariant feature transforms) features [11] (corresponding to the start/end points of the thin lines in Fig 3). Each *SIFT* feature identifies points of interest in the images. The main advantage of *SIFT* features is that they are easy to calculate. While each image contains a large number of features (typically around 1000) normally only a few (~ 70) will match with a prototype (thin lines in Fig. 3). On the other hand, if a consistent transformation (scaling/rotation/translation) of only a small set of features ($\sim 10-20$, thick lines in Fig. 3) between the image and the prototype can be calculated, the probability of a miss-classification is extremely low. To calculate the transformation we used the *RANSAC* (random consensus) algorithm [12], which automatically eliminates false matches (like in the hair region in Figure 3). The number of matched features is considered to be proportional to the probability of the card being a match.

The output generated by the image analysis is highly interesting in our context because it often results in confusions between similar cards. Typical confusions occur between cards of the same suit (e.g, the 9 versus the 10) as well as between numbered cards belonging to different suits. These confusions show that this is a realistic setting for probabilistic rule learning.

We used two concepts to test the algorithm. The first sequence contains the concept that states that the next card has to be red, that is,

```
trans(PrevPos,Curr) :- red(Curr).
```

Learning this concept took 45 seconds. ProbFOIL found this rule but also discovered an extra rule

```
trans(PrevPos,Curr) :- black(PrevPos).
```

The last rule is valid but only an artifact as it does not cover any example neither positive nor negative. It disappears, when the m of the m-estimates is set to zero.

The second concept to learn was that black cards have to be followed by odd cards and red cards by even cards. The correct rule-set consists therefore of the two rules:

`trans(PrevPos,Curr) :- black(PrevPos),odd(Curr)` and
`trans(PrevPos,Curr) :- red(PrevPos),even(Curr)`

Again, ProbFOIL learned some extra rules covering some very small noisy fractions of examples:

`trans(PrevPos,Curr) :- odd(Curr),even(Curr),red(Curr)` and
`trans(PrevPos,Curr) :- black(Curr),even(Curr),odd(Curr).`

The last two rules are logically inconsistent as cards cannot be even and odd at the same time, but due to the noise in the observations, they provide some extra accuracy. In any case it is interesting to see how the accuracy evolves as more rules are learned. This is graphically depicted in Figure 2 right. The accuracy of the rule stating that everything is positive is 0.56, the accuracy of the different rule-sets learned by ProbFOIL are in order of discovery 0.574, 0.671, 0.673, 0.673. This also implies that the target concept itself has only an accuracy of 0.671, and that the last two rules that are added only account for 0.2% accuracy. Thus the improvement of these last two rules is marginal and they would typically be removed should we employ a kind of post-pruning.

6 Conclusions

We have introduced a novel setting for probabilistic rule learning and developed the ProbFOIL algorithm for solving it using the probabilistic logic programming system ProbLog. The result is a natural probabilistic extension of inductive logic programming and rule learning. There are several remaining open questions that we are currently exploring. First, it would be interesting to experimentally compare the present approach to some alternative approaches (as sketched in the introduction). Secondly, we are developing an approach to probabilistic rule learning in which each rule contains an extra probabilistic predicate.

Acknowledgements

The authors thank Manfred Jaeger, Guy Van den Broeck, Peter Flach, and Wannes Meert for helpful discussions. This research was supported by the FWO project: Relational action and activity learning and the GOA/08/008 project Probabilistic Logic Learning.

References

1. Pan, R., Peng, Y., Ding, Z.: Belief update in bayesian networks using uncertain evidence. In: ICTAI 2006: Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence, pp. 441–444. IEEE Computer Society, Washington, DC, USA (2006)
2. Karalic, A., Bratko, I.: First order regression. Machine Learning 26(2-3), 147–176 (1997)
3. Srinivasan, A., Page, D., Camacho, R., King, R.D.: Quantitative pharmacophore models with inductive logic programming. Machine Learning 64(1-3), 65–90 (2006)
4. De Raedt, L., Kimmig, A., Toivonen, H.: Problog: A probabilistic Prolog and its application in link discovery. In: Veloso, M. (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 2462–2467 (2007)
5. Kimmig, A., Santos Costa, V., Rocha, R., Demoen, B., De Raedt, L.: On the efficient execution of probLog programs. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 175–189. Springer, Heidelberg (2008)
6. De Raedt, L.: Logical and Relational Learning. Springer, Heidelberg (2008)
7. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
8. Fürnkranz, J., Flach, P.A.: Roc 'n' rule learning — towards a better understanding of covering algorithms. Machine Learning 58(1), 39–77 (2005)
9. Fürnkranz, J., Widmer, G.: Incremental reduced error pruning. In: Cohen, W., Hirsh, H. (eds.) ICML 1994: Proceedings of the 11th International Conference on Machine Learning, pp. 70–77. Morgan Kaufmann, San Francisco (1994)
10. Dietterich, T., Michalski, R.: Learning to predict sequences. In: Michalski, R., Carbonell, J., Mitchell, T. (eds.) Machine learning: An Artificial Intelligence Approach, vol. 2. Morgan Kaufmann, San Francisco (1986)
11. Lowe, D.G.: Object recognition from local scale-invariant features. In: ICCV, pp. 1150–1157 (1999)
12. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM 24(6), 381–395 (1981)

Interactive Discriminative Mining of Chemical Fragments

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

[1] CRACS-INESC Porto LA, Universidade do Porto,
Rua do Campo Alegre 1021/1055, 4169-007 Porto, Portugal
[2] LIAAD-INESC Porto LA & DEI-FEUP, Universidade do Porto,
Rua Dr Roberto Frias s/n, 4200-465 Porto, Portugal
[3] CRACS-INESC Porto LA & DCC-FCUP, Universidade do Porto,
Rua do Campo Alegre 1021/1055, 4169-007 Porto, Portugal

Abstract. Structural activity prediction is one of the most important tasks in chemoinformatics. The goal is to predict a property of interest given structural data on a set of small compounds or drugs. Ideally, systems that address this task should not just be accurate, but they should also be able to identify an *interpretable* discriminative structure which describes the most discriminant structural elements with respect to some target.

The application of ILP in an interactive software for discriminative mining of chemical fragments is presented in this paper. In particular, it is described the coupling of an ILP system with a molecular visualisation software that allows a chemist to graphically control the search for interesting patterns in chemical fragments. Furthermore, we show how structural information, such as rings, functional groups such as carboxyls, amines, methyls, and esters, are integrated and exploited in the search.

Keywords: Drug design, graphical mining, efficiency.

1 Introduction

Structural activity prediction is one of the most important tasks in chemoinformatics. The goal is to predict a property of interest given structural data on a set of small compounds or drugs. This task can be seen as an instance of a more general task, *Structure-Activity Relationship* (SAR), where one aims at predicting the activity of a compound under certain conditions, given structural data on the compound. Ideally, systems that address this task should not just be accurate, they should be able to identify an *interpretable* discriminative structure which describes the most discriminant structural elements with respect to some target.

In an invited talk to Computational Logic 2000 and ILP'2000 David Page [1] highlighted the importance of interactive ILP systems for SAR problems. The application of Inductive Logic Programming (ILP) in an interactive software for discriminative mining of chemical fragments is presented in this paper. In particular, it is described a software application, called *iLogCHEM*, that allows a chemist to graphically control the search for interesting patterns in chemical fragments. *iLogCHEM* couples an ILP system with a molecular visualisation software, thus leveraging the flexibility of ILP while addressing the SAR task mentioned above. *iLogCHEM* can input data from chemical representations, such as MDL's SDF file format, and display molecules and matching patterns using visualisation tools such as VMD [2]. It has been demonstrated [3] that *iLogCHEM* can be used to mine effectively large chemoinformatics data sets, such as the DTP AIDS data set [4].

The focus of this paper is on allowing domain expert users to participate in the drug discovery process in a number of ways:

1. We propose the ability to incorporate user-provided abstractions of interest to the chemoinformatics domain, that can be used to aid the discovery process. As a first experiment, we have allowed users to specify a common chemical structure, *aromatic rings*. The user has available in *iLogCHEM*, apart from the *aromatic rings*, functional groups such as *carboxyl*, *amine*, *ester*, *methyl*, *phenyl* etc. This is supported through a *macro* mechanism (described in more detail in Section 4) where the user provides a pattern which is used to control rule refinement.
2. We propose an interactive refinement process where the user can interact with the proposed model, adapting it, evaluating it, and using it to guide (constrain) the search.
3. A common procedure in drug design is to introduce small variations in well known molecules. This procedure leads to data bases with groups of molecules that are very similar. When data sets are assembled from these data bases there is a "similarity bias". To attenuate that effect *iLogCHEM* allows the user to compute the similarity between the data set molecules and discard the more similar ones, retaining a set of "representative" ones. A more detailed description of this facility is described in Section 3.

The rest of the paper is organised as follows. Section 2 provides a brief introduction to the SAR problem and the issue of molecular representations. Section 3 introduces *iLogCHEM* and describes its main components. Section 4 describes its ability to incorporate user-provided abstractions of interest to the chemoinformatics domain through the use of what we have designated as *macros*. Section 5 explains the facilities for interactive search and refinement. Finally, conclusions and future work are described in Section 6.

2 Background

Structure activity relationships (SAR) describe empirically derived relationships between a molecule and its activity as a drug. In a typical SAR problem the goal is to construct a predictive theory relating the structure of a molecule to its activity given a set of molecules of known structure and activity.

A problem that one has to address is how to describe molecules. Coordinate-based representations usually operate by generating features from a molecule's 3D-structure [5]. The number of features of interest can grow very quickly, hence the problem that these systems need to address is how to select the most interesting features and build a classifier from them. Coordinate-free representations can use atom pair descriptors or just the atom-bond structure of the molecule. In the latter case, finding a discriminative component quite often reduces to the problem of finding a Maximum Common Substructure (MCS).

Exact MCS search in a molecule represented as a set of atoms and bonds can be seen as a graph-mining task. In this case, a molecule is represented as a graph $G_M = (V, E)$ where V, the vertices, are atom labels, and E, the edges, are bonds. The search can be improved by adding atom and bond properties. The earliest approaches to search for common substructures or fragments were based on ideas from Inductive Logic Programming (ILP). ILP techniques are very appealing because they are based on a very expressive representation language, first order logic, but they have been criticised for exhibiting significant efficiency problems. As stated by Karwath and De Raedt [6], "their application has been restricted to finding relatively small fragments in relatively small databases". Specialised graph miners have therefore become quite popular. Systems such as SUBDUE [7] started from the empty graph and then generate refinements either using beam-search or breadth-first search. More recent systems such as MoFa [8], gSpan [9], FFSM [10], Gaston [11], FMiner [12] and SMIREP [6], perform depth-first search, and use compact and efficient representations, such as SMILES, for which matching and canonical forms algorithms exist. Arguably, although such systems differ widely, they all use three main principles: **(i)** only refine fragments that appear in the database; **(ii)** filter duplicates; and **(iii)** perform efficient homomorphism testing.

3 The *iLogCHEM* System

iLogCHEM is an interactive tool for discriminative mining of chemical fragments. *iLogCHEM* uses a logic representation of the molecules, where atoms and bonds are *facts* stored in a database. Although our representation is less compact than a specialised representation such as SMILES, used in MOLFEA [13] and SMIREP [6], it offers a number of important advantages. First, it is possible to store information both on atoms and on their location: this is useful for interfacing with external tools. Second, *iLogCHEM* can take advantage of the large number of search algorithms implemented in ILP. Third, given that we implement the basic operations efficiently, we can now take advantage of the flexibility of our framework to implement structured information.

The interaction with the system is made through a graphical user interface. The system requires two input files: one is in SDF format with atom and bond data on a set of molecules; the other is a file which labels (discriminates) the compounds. We use SDF because it is highly popular and because it can convey 3D structure information. Other formats, such as SML can be translated to SDF through tools such as OpenBabel [14]. Also note that some datasets, such as the DSSTox [15] collection of datasets with at most 2000 molecules, include 2D and 3D information in the SDF format. Furthermore, the user may choose from 22 1D descriptors, 300 molecular fingerprints and 242 2D descriptors, predefined by chemists. These descriptors can be analysed with propositional tools, not just ILP.

The input files (in SDF format) are processed and given as input to a rule discovery algorithm, that is implemented as an extension of an ILP system (currently Aleph [16]). We significantly improved the ILP search algorithm for this task, as explained in the next section. The ILP engine allows the introduction of extra background knowledge for rule discovery. As an example, we take advantage of this flexibility by allowing the user to introduce well-known molecular structures in the search process. This is supported through a *macro* mechanism (described in more detail in Section 4) where the user provides a pattern which is used to control rule refinement.

The output of the ILP system will be a set of rules, or *theory*. Most often, chemists will be interested in looking at individual rules. *iLogCHEM* first matches the rules against the database, and then allows the user to navigate through the list of matches and visualise them. *iLogCHEM* uses `VMD` [2] to display the molecules and the matching substructures.

The key component of *iLogCHEM* is rule discovery. From a number of ILP algorithms, we chose to base our work on Progol's greedy cover algorithm with Mode Directed Inverse Entailment algorithm (MDIE) [17], as implemented in the Progol, April [18], and Aleph systems [16]. We rely on MDIE to achieve directed search, and we use greedy cover removal as a natural algorithm for finding interesting patterns. Figure 1 shows an example pattern for the HIV data set. The pattern is shown as a wider atoms and bonds, and it includes a sulphur atom and part of an aromatic ring.

Molecular Filtering. It is a common practise in drug design to take a small molecule that exhibits some activity and introduce small changes to it to improve its activity. This procedure produces a large set of similar molecules. Most of the available data for drug design suffer from this "similarity bias". Using *iLogCHEM* the similarities between any two pairs of molecules can be computed and retain only the "representative" ones, producing an unbiased data set. *iLogCHEM* uses Tanimoto distance to assess the similarity of two molecules. The user can specify a threshold value to be used in the filtering procedure.

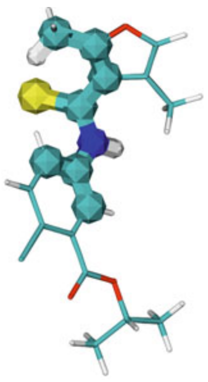


Fig. 1. HIV Pattern (wider atoms and bonds) discovered by ILP

Pattern Enumeration. *iLogCHEM* enumerates patterns (or sub-graphs) contained in an example molecule, the seed. To do so it uses the LogCHEM algorithm [3], based on the Aleph ILP system [16] to constrain the search space. This algorithm keeps a trie with previously generated clauses, according to a Morgan normal form, and tries to optimise rule evaluation for the specific domain of chemical compounds.

Pattern Matching. Given a new pattern, we are interested in finding out how many molecules support the pattern. ILP systems rely on refutation for this purpose. However, this introduces a problem. Consider the clause:

$$\begin{array}{l} active(C) \leftarrow \\ \qquad atom(C, Id_1, \mathtt{c}) \wedge \\ \qquad atom\_bond(C, Id_1, Id_2, \mathtt{c}, \mathtt{n}, 2) \wedge \\ \qquad atom\_bond(C, Id_1, Id_3, \mathtt{c}, \mathtt{n}, 2) \end{array}$$

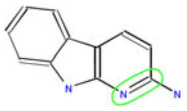


Fig. 2. An Example Pattern from a Small Organic Molecule: A-alpha-C

that represents a $N = C = N$ pattern. Figure 2 matches the molecule A-alpha-C against the pattern. Clearly, there is no match. Unfortunately, Prolog finds a match by matching the same nitrogen against the pattern *twice*. This problem, known as Object Identity [19], is addressed by dynamically rewriting the rules so that different variables match different atoms:

$$\begin{array}{l} active(C) \leftarrow \\ \qquad atom(C, Id_1, \mathtt{c}) \wedge \\ \qquad atom\_bond(C, Id_1, Id_2, \mathtt{c}, \mathtt{n}, 1) \wedge \\ \qquad Id_1 \neq Id_2 \wedge \\ \qquad atom\_bond(C, Id_1, Id_3, \mathtt{c}, \mathtt{n}, 1) \wedge \\ \qquad Id_1 \neq Id_3 \wedge Id_2 \neq Id_3 \end{array}$$

iLogCHEM includes a number of further optimisations. Namely, we rewrite bond information in such a way as to minimise backtracking. Also, by default, *iLogCHEM* compiles every pattern, instead of interpreting them, as usual in ILP [3].

4 Integrating Structural Information in the Search

The *iLogCHEM* system has the ability to integrate complementary information in the pattern search process. Our work was motivated by two observations. First, quite often chemists rely on well-known structures that are typically influential in the chemical properties of compounds. Second, global properties of the compound may be good indicators of activity.

4.1 Macros

A first step forward stems from observing Figure 1: does the pattern include part of the ring because only part of the ring matters or, as it is more natural from the chemists point of view, should we believe that the whole ring is in the pattern? Quite often discriminative miners will only include part of a ring because it is sufficient for classification purposes. But this may not be sufficient to validate the pattern.

The logical representation used in *iLogCHEM* makes it natural to support *macro* structures, such as rings used in MoFa [8] in a straightforward fashion. The next example shows such a description:

```
macro(M,(atom(A1,c), bond(A1,A2,_),
     atom(A2,c), bond(A2,A3,_),
     atom(A3,c), bond(A3,A4,_),
     atom(A4,c), bond(A4,A5,_),
     atom(A4,c), bond(A4,A5,_),
     atom(A5,c), bond(A5,A6,_),
     atom(A6,c), bond(A6,A1,_))).
```

Initial experiments with *iLogCHEM* show that using such macros results in similar accuracy, but returns *easier to interpret rules*.

iLogCHEM has available a library of functional groups that may be used as macros to speed up search and are very useful to improve understandability of the models. Some of the functional groups available include: aldehyde, amine, methyl, ester, ketone, hydroxyl, cyano, carboxylic acid, etc

4.2 Molecular Properties

One of the major benefits of ILP for SAR is its ability to combine very diverse sources of information. *iLogCHEM* allows the user to select chemical properties of interest for a compound, and combine them with pattern generation. Properties of interest are obtained through the graphical interface, and then passed on to the miner. In *iLogCHEM* the user may choose from a wide set of 1D molecular descriptors. As an example, consider this predicate clause for the CPDBAS dataset [15]:

```
active(A) :-
  logp(A,B), B =< -0.73333657,
  atom(A,C,c), atom_bond(A,C,D,c,c,4), atom_bond(A,D,E,c,o,4).
```

The constant -0.73333657 is obtained from a seed example by saturation. Experiments with NCTRER dataset show that this feature can be quite useful in complementing graph search.

5 Interactive Search and Refinement

After choosing the data set of molecules and then filtering them out using Tanimoto distance the user may launch the ILP and obtain a first model. The user may then choose to visualise each rule of the model and overlap a rule (pattern) on the structure of a molecule covered by that rule (as shown in Figure 1).

Once a model is constructed there are two possible interactions the user can take. The user may decide to do a “local and manual” search or he can specify constraints on the visualised pattern and ask the ILP system to produce a new model.

In the first case the user may incrementally produce changes in the pattern (adding or deleting atoms and/or bonds) and then ask *iLogCHEM* to immediately evaluate the changed pattern. Whenever an evaluation is done the user can see a list of the “positive and negative” molecules covered.

If “local and manual” search does not produce the desired results the user may interactively (again adding/removing atoms and bonds) define a new pattern. This new pattern can be converted into a clause and used as the starting clause of the search space. That is, the user is commanding the system to find useful refinements of the provided pattern.

6 Conclusions

This paper reports on *iLogCHEM*, an interactive tool to be used in interactive drug design tasks. *iLogCHEM* is designed to give users who have little knowledge of, or interest in ILP the benefits of this learning mechanism. Thus, it can be seen as step forward in "enhancing human–computer interaction to make ILP systems true collaborators with human experts" [20].

iLogCHEM is founded on previous work to create an effective rule miner for ILP [3]. The system was driven by expert requirements, extending the previous work as follows by introducing i) a library of preexisting common patterns, considered relevant by experts, that are immediately available for discovery; and ii) the ability to define a new pattern *graphically* and then translate it to the *iLogCHEM* internal representation. These new facilities enable the expert to: i) look at the pattern highlighted on the molecule structure; ii) interact with the visualisation tool and specify constraints not satisfied by the pattern presented; and iii) rerun the ILP system with the specified constraints added to the data set. These steps are the centre of the main loop of the interaction where the expert guides the process of pattern discovery. Additionally the tool also allows the expert user to specify a list of chemical structures (rings and functional groups) that are used as macro operators. The use of chemical structures may be very useful to achieve more compact and comprehensible models than the ones described with atoms and bonds.

Acknowledgements

This work has been partially supported by the project ILP-Web-Service (PTDC-/EIA/70841/2006), HORUS (PTDC/EIA-EIA/100897/2008), and by the Fundação para a Ciência e Tecnologia. Max Pereira is funded by FCT grant SFRH/-BPD/37087/2007.

References

1. Page, D.L.: ILP: Just do it. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 3–18. Springer, Heidelberg (2000)
2. Humphrey, W., Dalke, A., Schulten, K.: VMD – Visual Molecular Dynamics. Journal of Molecular Graphics 14, 33–38 (1996)
3. Costa, V.S., Fonseca, N.A., Camacho, R.: LogCHEM: Interactive Discriminative Mining of Chemical Structure. In: Proceedings of 2008 IEEE International Conference on Bioinformatics and Biomedicine (BIBM 2008), pp. 421–426. IEEE Computer Society, Philadelphia (2008)
4. Collins, J.M.: The DTP AIDS antiviral screen program (1999), `http://dtp.nci.nih.gov/docs/aids/aids_data.html`
5. Maggiora, G.M., Shanmugasundaram, V., Lajiness, M.J., Doman, T.N., Schultz, M.W.: A practical strategy for directed compound acquisition, pp. 315–332. Wiley-VCH, Chichester (2004)

6. Karwath, A., De Raedt, L.: Predictive Graph Mining. In: Suzuki, E., Arikawa, S. (eds.) DS 2004. LNCS (LNAI), vol. 3245, pp. 1–15. Springer, Heidelberg (2004)
7. Chittimoori, R.N., Holder, L.B., Cook, D.J.: Holder, and Diane J. Cook. Applying the subdue substructure discovery system to the chemical toxicity domain. In: Kumar, A.N., Russell, I. (eds.) Proceedings of the Twelfth International Florida Artificial Intelligence Research Society Conference, Orlando, Florida, USA, May 1-5, pp. 90–94. AAAI Press, Menlo Park (1999)
8. Borgelt, C., Berthold, M.R.: Mining molecular fragments: Finding relevant substructures of molecules. In: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), Japan, pp. 51–58 (2002)
9. Yan, X., Han, J.: gspan: Graph-based substructure pattern mining. In: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), Maebashi City, Japan, December 9-12 (2002)
10. Huan, J., Wang, W., Prins, J.: Efficient mining of frequent subgraphs in the presence of isomorphism. In: Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), Melbourne, Florida, USA, December 19-22, pp. 549–552. IEEE Computer Society, Los Alamitos (2003)
11. Nijssen, S., Kok, J.N.: Frequent graph mining and its application to molecular databases. In: Proceedings of the IEEE International Conference on Systems, Man & Cybernetics, The Hague, Netherlands, October 10-13, pp. 4571–4577. IEEE, Los Alamitos (2004)
12. Maunz, A., Helma, C., Kramer, S.: Large-scale graph mining using backbone refinement classes. In: KDD, pp. 617–626 (2009)
13. Kramer, S., De Raedt, L., Helma, C.: Molecular feature mining in hiv data. In: KDD, NY, USA, pp. 136–143 (2001)
14. Guha, R., Howard, M.T., Hutchison, G.R., Murray-Rust, P., Rzepa, H., Steinbeck, C., Wegner, J.K., Willighagen, E.L.: The Blue Obelisk–Interoperability in Chemical Informatics. Journal of Chemical Information and Modeling 46, 991–998 (2006)
15. Richard, A.M., Williams, C.R.: Distributed structure-searchable toxicity (dsstox) public database network: a proposal. Mutation Research/Fundamental and Molecular Mechanisms of Mutagenesis 499, 27–52(26) (2002)
16. Srinivasan, A.: The Aleph Manual. University of Oxford (2004), http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/
17. Muggleton, S.: Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming 13(3-4), 245–286 (1995)
18. Fonseca, N.A., Silva, F., Camacho, R.: April – An Inductive Logic Programming System. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, pp. 481–484. Springer, Heidelberg (2006)
19. Lisi, F.A., Ferilli, S., Fanizzi, N.: Object identity as search bias for pattern spaces. In: van Harmelen, F. (ed.) Proceedings of the 15th Eureopean Conference on Artificial Intelligence, ECAI 2002, pp. 375–379. IOS Press, Amsterdam (2002)
20. Page, D., Srinivasan, A.: ILP: A short look back and a longer look forward. Journal of Machine Learning Research 4, 415–430 (2003)

MMRF for Proteome Annotation Applied to Human Protein Disease Prediction

Beatriz García-Jiménez, Agapito Ledezma, and Araceli Sanchis

Universidad Carlos III de Madrid
Av.Universidad 30, 28911, Leganés, Madrid, Spain
beatrizg@inf.uc3m.es

Abstract. Biological processes where every gene and protein participates is an essential knowledge for designing disease treatments. Nowadays, these annotations are still unknown for many genes and proteins. Since making annotations from in-vivo experiments is costly, computational predictors are needed for different kinds of annotation such as metabolic pathway, interaction network, protein family, tissue, disease and so on. Biological data has an intrinsic relational structure, including genes and proteins, which can be grouped by many criteria. This hinders the possibility of finding good hypotheses when attribute-value representation is used. Hence, we propose the generic Modular Multi-Relational Framework (MMRF) to predict different kinds of gene and protein annotation using Relational Data Mining (RDM). The specific MMRF application to annotate human protein with diseases verifies that group knowledge (mainly protein-protein interaction pairs) improves the prediction, particularly doubling the area under the precision-recall curve.

Keywords: Relational Data Mining, Human Disease Annotation, Multi-Class Relational Decision Tree, First-Order Logic, Structured Data.

1 Introduction

Functional annotation consists of attaching biological information to gene and genetic product sequences. For instance, identifying whether a gene is involved in a biological process, a regulation network or a molecular function; or assigning to a protein its expression profile or phenotype (tissue or disease association). Knowing the processes in which genes and proteins are involved is an essential knowledge to design disease treatments.

Nowadays, a gene/protein appears annotated in multiple distributed repositories. However, many proteins have still few or no annotation in a large number of species, because experimental techniques are costly in resources and time. This process is also overwhelmed by the high amount of data that need to be acquired and managed. Therefore, computational prediction methods have shown an useful alternative in the last years [16], in order to focus the experimental verifications on the hypotheses (predicted annotations) that are more likely to be true.

Many diverse prediction techniques have been proposed to solve the genome[1] annotation problem. Each method uses different kind and amount of input data, and is focused on a particular prediction goal. This variability in methods makes difficult a comparison among them. The simplest prediction approach is based on sequence similary, as Blast2GO [1], only useful for Gene Ontology (GO) annotation. Others predictors just include sequence and structure features [13]; while more sophisticated methods integrate heterogeneous data sources, such as Fatigo [1] and DAVID [5]. Some techniques simplify the data representation to numerical features, applying subsymbolic machine learning algorithms [10,12]; but others preserve the intrinsic structure of biological data, applying Multi-Relational Data Mining (MRDM). These techniques take advantages of the interpretable symbolic representation, such as [4,7,21] in functional annotation and [19,20] in other related bioinformatic domains.

Despite all these efforts, the proteome annotation problem remains open. We do not know functions and tasks for all proteins, and many annotations are neither verified by experts nor complete in all the biological fields of interest. Particularly, there are few specialized predictors in disease annotation, being an essential knowledge to design new drugs. Morbid OMIM (Online Mendelian Inheritance in Man) [2] contains information on all known mendelian disorders and associated genes. It is the most complete and updated repository about genetic disorders. This repository is carefully curated and frequently referenced by biological and medical scientists. For these reasons, we decide to use OMIM instead of other less known disease vocabulary such as eVOC pathology [11]. Most annotation methods using OMIM perform search rather than prediction. Some approaches predict new annotation [14], but none applies MRDM.

To summarize, genome and proteome annotation prediction is still an open problem with regard to various kind of specific annotation. Disease annotation is one of special interest. This paper proposes applying MRDM to a relevant annotation domain: human disease prediction. Besides, we want to verify the relevance of biological group relations using data integrated from different data sources. This group data is very suitable to be exploited by relational learning. We address this problem adapting a generic and flexible framework, MMRF [6], which can easily predict different annotations.

Several facts support this proposal. First, MRDM have been succesfully applied to other related bioinformatic domains. Second, we use up to date data from different biological databases used in many current science projects. Finally, we have made a special effort in data collection, selecting only experimental data, when it is possible, in order to avoid indirect redundancies coming from internal predictions from other applications, which can bias the results.

This paper is organized as follows: Section 2 briefly explains the Modular Multi-Relational Framework. The human protein OMIM disease prediction domain is described in Section 3. Section 4 presents and analyzes the application results. Finally, in Section 5, conclusions and future work are summarized.

[1] In annotation context, the terms *gene* and *protein* or *genome* and *proteome* are indistinctly used.

2 Modular Multi-Relational Framework

Modular Multi-Relational Framework (MMRF) [6] is a system originally designed for gene *Group* function prediction domain, facing the problem from a relational and flexible point of view. It has been applied to predict function for *S.cerevisiae* (i.e.Yeast) genes grouped by complexes [7]. Now, we adapt the framework since we have realized that group annotation problem is very complex to face in a single step [7]. The changes aim to solve gene and protein *individual* function annotation prediction problem, instead of *group* annotation. Nevertheless, this MMRF layout can also be considered the first phase for the group annotation problem. The complete process could be achieved obtaining first annotations for individual group elements using MMRF, and then combining them for group annotations using an alternative method (for example, union or intersection of individual annotations).

MMRF preserves the same main properties as the original layout. It is designed by modules for managing the high variability that the functional annotation biological domain entails. This facilitates changing independiently data, criteria and methodology. MMRF uses a multi-relational approach (in representation and learning) for fitting the intrinsic relational structure of gene and protein group data, and for integrating different data sources.

Figure 1 shows the new MMRF layout oriented to individual protein annotation prediction. Module 2 is now called *Selecting annotation* where the annotation vocabulary is chosen and assigned to individual gene or protein. The relational knowledge about belonging to specific biological groups (i.e.metabolic pathways, regulation networks, etc.[6,7]) is handled in module 3.

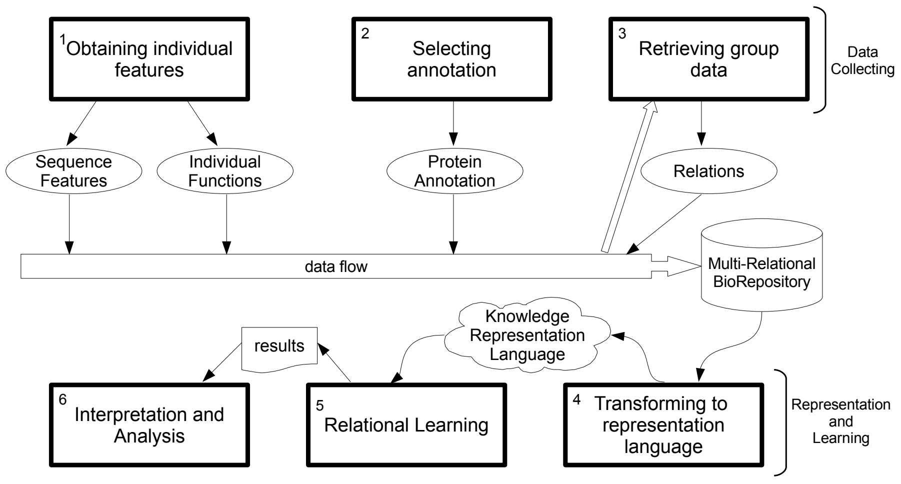

1 Obtaining individual features
2 Selecting annotation
3 Retrieving group data
Data Collecting
Sequence Features
Individual Functions
Protein Annotation
Relations
data flow
Multi-Relational BioRepository
Knowledge Representation Language
results
6 Interpretation and Analysis
5 Relational Learning
4 Transforming to representation language
Representation and Learning

Fig. 1. A new schema of the Modular Multi-Relational Framework (MMRF). The rectangles represent modules and the ellipses represent data.

3 MMRF Applied to Human Protein Disease Prediction

This section describes the MMRF module instantiations for applying the framework to the Human Protein OMIM Disease Prediction domain.

1.Obtaining individual features. In this application, there are 7 features derived from gene sequence (such as chromosome name and length, or transcript count) and 27 features from protein sequence (including length, positive and negative charge from the sequence, aminoacid composition and whether the protein contains a transmembrane, signal or coiled-coil domain). Also, 5 different kinds of protein functional annotations are collected, related with protein family (from *Pfam*), protein domain (from *InterPro*), biological process, cellular component and molecular function. The last three come from *Gene Ontology (GO)*, and are only from experimental results, ignoring automated annotation, for avoiding biases induced by overlaps with others annotation sources.

The numerical protein sequence features are computed with BioWeka [8] using as input the *UniProt* aminoacid sequences in FASTA format. Only *Swiss-Prot* sequences are included, because the remainder (*TrEMBL* sequences) have not been reviewed by experts. The rest of features are retrieved from *Ensembl* project, through the BioMart tool [17]. See the module 1 instantiation schema in Supplementary Material.

2.Selecting annotation. The annotation goal is genetic disorders using gene-disease associations from Morbid OMIM [2]. We apply a manual OMIM disorder categorization made by experts [9]. These disease categories have been recently used in other studies [14]. Thus, the 4,927 OMIM disorders [2] are categorized in 23 disease classes based on the affected physiological system. Some of the classes are: neurological, cancer, cardiovascular, inmunological or endocrine disease (see a complete list in Supplementary Material). Therefore, this MMRF application classifies proteins in these general disease categories [9]. However, a simple modification in MMRF module 2 could easily build a particular predictor for diseases at lower level, for instance, knowing in which specific kind of cancer (leukemia, melanoma, breast cancer, etc.) a protein is involved.

3.Retrieving group data. Two sources of group data are included, though it could be easily increased with others, as protein complexes or co-expresion data. The first data source consist of protein-protein interaction pairs, retrieved from BioGRID repository (2.0.59 Release) [18], which integrates important interaction databases as MINT, IntAct or HPRD. We select BioGRID pairs from real binary relations identified by evidences codes *Co-crystal structure, Far Western, FRET, PCA* and *Two-Hybrid.* These interactions do not include pairs split off from N-ary relations. It results in 21,687 proteins with 229,407 interactions among them. The second data source comprises metabolic pathways, which correspond to the 52 top-level human Reactome [15] pathways including 5,128 proteins, on average 159.85 proteins per pathway. See the module 3 instantiation schema in Supplementary Material.

[2] From OMIM Morbid Map on November 17th, 2009.

Data sources collected in modules 1 to 3 use different gene or protein identifiers. The original identifiers are all mapped to Ensembl (gene or protein) IDs using the cross-references from BioMart [17] queries.

4.Transforming to representation language. The knowledge representation language is a subset of first-order logic. All the collected data previously described is represented as predicates in Prolog syntax (see Figure 2). Since for humans, we can not assume the simplification *1-gene:1-protein*, as simpler organism does, the representation language has to handle information level with regard to gene and protein. These levels are related by the *1-gene:N-proteins* relation (represented as N binary predicates `protein_gene/2` per gene). Thus, the different features are separately associated to genes (predicates with *geneID* as key) or to proteins (predicates with *protID* as key) (see Figure 2). Moreover, the group data has a different representation depending on the number of elements in the group. Binary relations are represented as pairs (i.e. `ppinteracion_pair/2`). N-ary relations are represented with one group identifier plus N binary predicates (i.e. `protein_in_pathway/2`), where each predicate relates a group element with the group identifier.

```
gene(geneID,name,length,strand,trCount).            gene_biotype(geneID,bioType).
protein(protID,length,posCharge,negCharge).         protein_class(protID,omimID).
aa_composition(protID,aaID,proportionAA).           protein_gene(protID,geneID).
go_annotation_bioProcess(protID,goID).              transmembrane_domain(protID).
ppinteraction_pair(protID,protID).                  ncoils_domain(protID).
protein_in_pathway(groupID,protID).                 pfam_domain(protID,pfamID).
...
```

Fig. 2. Fragment of the knowledge representation language in proteome disease prediction domain

The instantiation of module ***5.Relational Learning*** consist of applying the algorithm TILDE [3], implemented in the ACE tool, using a multi-class and multi-label learning, inspired by other works [21]. The instantiation of module ***6.Interpretation and Analysis*** consist of evaluating the result with Precision-Recall curves (PRC). For more details, see a previous MMRF application [7], which shares the same instantiations of modules 5 and 6.

4 Results and Discussion

This section describes the results of predicting human protein diseases with MMRF. The whole data set comprises 6,958 protein-disease annotations, for 5,640 different proteins (examples) and 21 diseases (classes). Each protein can be associated with more than one disease, ranged from 1 to 5, in this set. On average, there are 331.3 annotations per disease. There are at least 40 proteins per class (the two classes with less than this minimum have been ignored). On average, there are around 5% positive vs 95% negative examples per class, although the protein class distribution is not equitable (see Supplementary Material). Each of the four majority classes has more than 10% of all annotations.

The background knowledge also includes related proteins without disease associations, but belonging to a metabolic pathway or having an interaction with a disease protein.

We compare two configurations, which differ in module 3 instantiation, it means on group relational data used for learning. The first one (***a.- Without groups***) does not included neither pathway nor protein-protein interaction data. The second configuration (***b.- With groups***) includes both kind of data from biological groups. In addition, we analize the learning implications of relational knowledge representation for groups.

The results shown in Table 1 and Figure 3 come from three folds cross validation experiments. Table 1 shows several quantitative measures and Precision-Recall curves appear in Figure 3 for the two configurations. All of them are the average results about overall 21 classes.

Table 1. Quantitative results from human protein disease prediction with MMRF. AU(PRC): Area Under Precision-Recall Curve. MSE: Mean Squared Error.

	Relational knowledge	
	a) without groups	**b) with groups**
AU(PRC):	0.282	0.625
Correlation:	0.290	0.599
MSE:	0.049	0.034

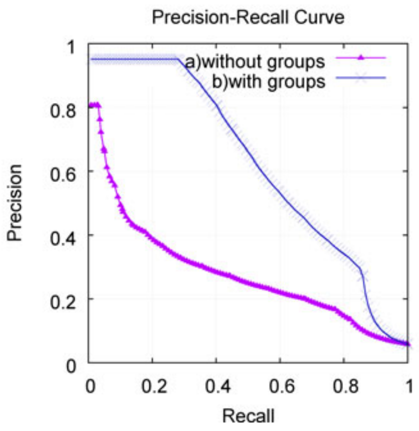


Fig. 3. Precision-Recall curves from human protein disease prediction with MMRF, in different configurations

Table 1 and Figure 3 point out that prediction with group data (configuration "b", on the right) improves the double upon without groups (configuration "a", on the left), in both AU(PRC) and correlation. Hence, this comparison asserts that knowledge about proteins belonging to a biological group is very relevant in disease annotation prediction.

Figure 4 presents a fragment of a relational decision tree from configuration with group data (b). The first tree node (see line 3) determines that `ppinteraction_pair/2` (a protein-protein interaction relation) is the most discriminative predicate. This fact confirms the relevance of group knowledge to predict annotations. Moreover, in the first 'yes'-branch (line 4), the second node includes a typical feature in protein function prediction: the positive charge of protein sequence [12] (variable Y), partially supporting the model reliability. Besides, in the first 'no'-branch (line 11), the most relevant query includes a *N:1* relation (predicate `protein_gene/2` relates a protein with the gene it comes from), emphasizing the high influence of relational knowledge on the prediction.

```
class(-A,-B,-C,-D,-E,-F,-G,-H,-I,-J,-K,-L,-M,-N,-O,-P,-Q,-R,-S,-T,-U,-V)
[0.011436170212766] 3760.0
ppinteraction_pair(A,-W),not(W=A) ?
+--yes: [0.0188476036618201] 1857.0
|       protein(W,-X,-Y,-Z),Y>=0.107506 ?
|       +--yes: [0.0219123505976096] 1004.0
|       |       ppinteraction_pair(W,W) ?
|       |       +--yes: [0.0569948186528497] 386.0
|       |       |       transmembrane_domain(W) ?
...
+--no:  [0.00420388859695218] 1903.0
        protein_gene(A,-M26),gene(M26,-N26,-O26,-P26,-Q26),O26>=84418 ?
        +--yes: [0.00981996726677578] 611.0
        ...
```

Fig. 4. Fragment of a relation decision tree in configuration with groups

Therefore, the importance of protein-protein interaction and protein-gene relations indicates that Relational Data Mining is essential in this domain. This is because to propositionalize this kind of data would be very complex or resulting in having redundant data. For instance, for protein binary relations, the single attribute-value table should have thousands of Boolean attributes, one per each protein. In addition, it should repeat all the gene features as attributes for all proteins that come from the same gene. Furthermore, attribute-value learning can not represent knowledge or retrieve hypotheses about features of related genes and proteins, as tree fragment in Figure 4 shows.

5 Conclusions and Further Work

This work highlights the relevance of biological group data for annotation prediction, particularly in proteome disease association. Since the most efficient and viable representation of this group knowledge is with relations, relational learning and the Modular Multi-Relational Framework are confirmed as very suitable for solving the proteome annotation problem. This is particularly relevant since the data comes from the integration of multiple up to date biological databases. Besides, the hypotheses learned through Relational Data Mining are mostly unfeasible to achieve in attribute-value learning and it holds the advantage of being readable for biology experts. This work has two main differences from a previous MMRF application [7]. For the annotation goal, diseases from OMIM morbid are used instead of general functions of GO Slim. Moreover the organism has been changed from yeast to human, which is more complex but more interesting. Thus, the obtained predictor let us select a subset of unknown protein-disease association (the most likely predictions) to be verified by in-vivo experiments.

As further work, many alternatives appear. It would be interesting to make a comparison between this overall classes predictor and 21 independent predictors, one per each disease class. Other possibilities would be related to biological MMRF applications. For instance, including new group data, such as protein complexes or co-expression data; applying the predictor to annotate unknown proteins; or changing the prediction goal to a different annotation field, as predicting if a protein belongs to a metabolic pathway.

Acknowledgments. This research has been supported by CICYT, TRA 2007-67374-C02-02 project and by the expert biological knowledge of the Structural Computational Biology Group in Spanish National Cancer Research Centre (CNIO). The authors would like to thank members of TILDE tool developer group in K.U.Leuven for providing their help and many useful suggestions.

Supplementary Materials (for online version). They include two schemas about obtaining individual features and retrieving group data, the complete list of 23 disease categories and a figure showing the protein per disease distribution.

References

1. Al-Shahrour, F., et al.: Babelomics: a systems biology perspective in the functional annotation of genome-scale experiments. Nucl. Acids Res. 34, W472–W476 (2006)
2. Amberger, J., et al.: McKusick's Online Mendelian Inheritance in Man (OMIM(R)). Nucl. Acids Res. 37, D793–D796 (2009)
3. Blockeel, H., De Raedt, L.: Top-down induction of logical decision trees. Artificial Intelligence 101(1-2), 285–297 (1998)
4. Clare, A.: Machine learning and data mining for yeast functional genomics. PhD thesis, University of Wales, Aberystwyth (2003)
5. Dennis, G., et al.: DAVID: Database for Annotation, Visualization, and Integrated Discovery. Genome Biology 4(5), P3 (2003)
6. García, B., et al.: Modular Multi-Relational Framework for Gene Group Function Prediction.. In: Online Proceedings ILP (2009)
7. García Jiménez, B., Ledezma, A., Sanchis, A.: S.cerevisiae complex function prediction with modular multi-relational framework. In: García-Pedrajas, N., Herrera, F., Fyfe, C., Benítez, J.M., Ali, M. (eds.) IEA/AIE 2010. LNCS, vol. 6098, pp. 82–91. Springer, Heidelberg (2010)
8. Gewehr, J., et al.: BioWeka extending the Weka framework for bioinformatics. Bioinformatics 23(5), 651–653 (2007)
9. Goh, K., et al.: The human disease network. PNAS 104(21), 8685–8690 (2007)
10. Jensen, J., et al.: Prediction of human protein function according to Gene Ontology categories. Bioinformatics 19(5), 635–642 (2003)
11. Kelso, J., et al.: eVOC: A Controlled Vocabulary for Unifying Gene Expression Data. Genome Research 13(6a), 1222–1230 (2003)
12. Lee, B., et al.: Identification of protein functions using a machine-learning approach based on sequence-derived properties. Proteome Science 7(1), 27 (2009)
13. Lee, D., et al.: Predicting protein function from sequence and structure. Nature reviews. Molecular Cell Biology 8(12), 995–1005 (2007)
14. Linghu, B., et al.: Genome-wide prioritization of disease genes and identification of disease-disease associations from an integrated human functional linkage network. Genome Biology 10(9), R91 (2009)
15. Matthews, L., et al.: Reactome knowledgebase of human biological pathways and processes. Nucl. Acids Res. 37, D619–D622 (2009)

16. Peña-Castillo, L., et al.: A critical assessment of mus musculus gene function prediction using integrated genomic evidence. Genome Biology 9, S2 (2008)
17. Smedley, D., et al.: BioMart-biological queries made easy. BMC Genomics 10 (2009)
18. Stark, C., et al.: BioGRID: a general repository for interaction datasets. Nucl. Acids Res. 34, 535–539 (2006)
19. Trajkovski, I., et al.: Learning relational descriptions of differentially expressed gene groups. IEEE Transactions on Systems, Man, and Cybernetics 38(1), 16–25 (2008)
20. Tran, T.N., Satou, K., Ho, T.-B.: Using inductive logic programming for predicting protein-protein interactions from multiple genomic data. In: Jorge, A.M., Torgo, L., Brazdil, P.B., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS (LNAI), vol. 3721, pp. 321–330. Springer, Heidelberg (2005), http://dx.doi.org/10.1007/11564126_33
21. Vens, C., et al.: Decision trees for hierarchical multi-label classification. Machine Learning 73(2), 185–214 (2008)

Extending ProbLog with Continuous Distributions

Bernd Gutmann[1], Manfred Jaeger[2], and Luc De Raedt[1]

[1] Department of Computer Science, Katholieke Universiteit Leuven, Belgium
{bernd.gutmann,luc.deraedt}@cs.kuleuven.be
[2] Department of Computer Science, Aalborg University, Denmark
jaeger@cs.aau.dk

Abstract. ProbLog is a recently introduced probabilistic extension of Prolog. The key contribution of this paper is that we extend ProbLog with abilities to specify continuous distributions and that we show how ProbLog's exact inference mechanism can be modified to cope with such distributions. The resulting inference engine combines an interval calculus with a dynamic discretization algorithm into an effective solver.

1 Introduction

Continuous distributions are needed in many applications for building a natural model. Probabilistic logic programming languages, such as ProbLog and CP-Logic [1], have, so far, largely focused on modeling discrete distributions and typically perform exact inference. The PRISM [2] system provides primitives for Gaussian distributions but requires the exclusive explanation property which complicates modeling. On the other hand, many of the functional probabilistic programming languages, such as BLOG [3] and Church [4], can cope with continuous distributions but only perform approximate inference by a Markov Chain Monte Carlo approach. Typical statistical relational learning systems such as Markov Logic and Bayesian Logic Programs have also been extended with continuous distributions. The key contribution of this paper is a simple probabilistic extension of Prolog based on the distribution semantics with both discrete and continuous distributions. This is realized by introducing a novel type of probabilistic fact where arguments of the fact can be distributed according to a continuous distribution. Queries can then be posed about the probability that the resulting arguments fall into specific intervals. We introduce the semantics of using continuous distributions in combination with comparison operations and show how ProbLog's inference mechanism, based on Binary Decision Diagrams (BDDs) [5], can be extended to cope with these distributions. The resulting language is called Hybrid ProbLog.

Similarly to Hybrid ProbLog, Hybrid Markov Logic Networks (HMLNs) [6] aim at integrating Boolean and numerical random variables in a probabilistic-logic modeling framework. The kind of modeling supported by HMLNs is quite different in nature from the kind of modeling for which Hybrid ProbLog is designed. In an HMLN, one defines equations that function as soft constraints for relationships among numerical and logical variables. For example, one could express that the temperature on day d is typically around 20° Celsius using the weighted equality w `temperature`$(d) = 20$, where larger weights w lead to a larger penalty for deviations of *temperature*(d) from 20. All weighted formulae containing *temperature*(d), together, implicitly define a probability distribution for the random variable *temperature*(d) due to HMLN semantics. However, one cannot directly specify this distribution to be Gaussian with, for example, mean 20 and standard deviation 5. No exact inference methods have been developed for HMLNs.

It is also instructive to compare Hybrid ProbLog with Hybrid Bayesian Networks [7]. Apart from one being a logical-relational, and the other a purely propositional framework, there is also a key difference in the interaction between continuous and discrete random variables permitted. In Hybrid Bayesian Networks, the distributions of continuous variables (usually Gaussian) are typically conditioned on discrete variables, but continuous variables cannot be parents of discrete ones. In Hybrid ProbLog this order is reversed: continuous variables are at the roots of the directed model, and the discrete (Boolean) variables are conditioned on the continuous ones. Thus, Hybrid ProbLog provides modeling capabilities and exact inference procedures that, for the propositional case, are in some sense complementary to Hybrid Bayesian networks.

This paper has three main contributions. (1) An extension of ProbLog with continuous distributions, (2) formal study of its semantics, and (3) an efficient inference algorithm based on dynamic discretization.

The rest of this paper is organized as follows. Section 2 reviews basic concepts from ProbLog. Section 3 introduces the syntax and semantics of Hybrid ProbLog. Section 4 describes our exact inference algorithm. Before concluding, we evaluate the algorithm in Section 5.

2 ProbLog

ProbLog [8] is a recent probabilistic extension of Prolog. A ProbLog theory $T = F \cup \mathcal{BK}$ consists of a set of labeled facts $F = \{p_1 :: f_1, \cdots, p_n :: f_n\}$ and a set of definite clauses $\mathcal{BK}$ that express the background knowledge. The facts $p_j :: f_j$ in F are annotated with a probability p_j stating that $f_j\theta$ is true with probability p_j for all substitutions θ grounding f_j. The resulting facts $f_j\theta$ are called atomic choices and represent random variables; they are assumed to be mutually independent. It is not allowed to use a probabilistic fact in the heads of clauses in $\mathcal{BK}$. Let $\Theta = \{\theta_{j1}, \ldots \theta_{ji_j} | j = 1, \ldots n\}$ be a finite[1] set of possible substitutions for the variables in the probabilistic facts where i_j is the number of substitutions for fact j, then a ProbLog theory describes a probability distribution over Prolog programs $L \subseteq L_F$ where $L_F = F\Theta$ and $F\Theta$ denotes the set of all possible ground instances of facts in F:

$$P_P(L|F) := \prod_{f_j\theta_{jk} \in L} p_j \prod_{f_j\theta_{jk} \in L_F \setminus L} (1 - p_j) \ . \tag{1}$$

[1] Throughout the paper, we assume that $F\Theta$ is finite, but see [2] for the infinite case.

The *success probability* of a query q then is

$$P_s(q|T) := \sum_{\substack{L \subseteq L_F: \\ L \cup \mathcal{BK} \models q}} P(L|F) \ . \tag{2}$$

ProbLog also defines a probability distribution P_w over possible worlds, that is Herbrand interpretations. Each total choice $L \subseteq L_F$ can be extended to a possible world by computing the least Herbrand model of L. This possible world is assigned the probability $P_w = P(L|F)$. Thus a set of total choices represents an assignment of truth-values to all atomic choices.

3 Hybrid ProbLog

A Hybrid ProbLog theory $T = F \cup F^c \cup \mathcal{BK}$ is a ProbLog theory extended by a set of continuous probabilistic facts[2] of the form

$$F^c = \{(X_1, \phi_1) :: f_1^c, \cdots, (X_m, \phi_m) :: f_m^c\}$$

where X_i is a Prolog variable, appearing in the atom f_i^c and ϕ_i is a density function. The fact $(\texttt{X}, \texttt{gaussian}(2, 8)) :: \texttt{temp}(\texttt{D}, \texttt{X})$, for example, states that the temperature for day `D` is Gaussian-distributed with mean 2 and standard deviation 8. The syntax allows one to specify multivariate distributions, i.e.,

$$((\texttt{X}, \texttt{Y}), \texttt{gaussian}([1, 0], [[1, 0.5], [0.5, 1]]) :: \texttt{f}(\texttt{X}, \texttt{Y}) \ .$$

In this paper, however, we restrict ourselves to the univariate case. From a user's perspective, continuous facts are queried like normal Prolog facts, and the value of the continuous variable is instantiated with a value drawn from the underlying distribution. Hybrid ProbLog adds the following predicates to the background knowledge of the theory to process values stemming from continuous facts:

- `below`(`X`,c) succeeds if `X` is a value from a continuous fact, c is a number constant, and $X < c$
- `above`(`X`,c) succeeds if `X` is a value from a continuous fact, c is a number constant, and $\texttt{X} > c$
- `ininterval`(`X`,c_1,c_2) succeeds if `X` is a value from a continuous fact, c_1 and c_2 are number constants, and $\texttt{X} \in [c_1, c_2]$

Unification with number constants is not supported for the values of continuous facts, i.e. the call `temp(d, 0)` fails. But one can express the same using

$$\texttt{temp}(\texttt{d}, \texttt{T}),\ \texttt{ininterval}(\texttt{T}, 0, 0) \ .$$

Similarly, standard Prolog comparison operators are not supported and one has to use the corresponding comparison predicate from the background knowledge:

$$\texttt{temp}(\texttt{d1}, \texttt{T}),\ \texttt{T} > 5$$

[2] We denote facts, values, and substitutions related to the continuous part of T by the superscript ^{c}.

has to be written as

$$\texttt{temp(d1,T), above(T,5)} .$$

Arithmetic expressions are not supported, i.e the query

$$\texttt{temp(d1,T), Fahrenheit is } 9/5 * \texttt{X} + 32, \ \texttt{Fahrenheit} > 41$$

is illegal. There is no equivalent predicate for that in the background knowledge. Also, the comparison of two continuous facts is not supported, i.e. the query

$$\texttt{temp(d1,T1), temp(d2,T2), above(T1,T2)}$$

is illegal. The latter restriction, in particular, prevents two or more continuous variables getting "coupled", i.e. there is a dependency that requires one to always consider the values of both variables simultaneously. Furthermore, disallowing arithmetic expressions ensures that one can partition the underlying $\mathbb{R}^n$ space into simple intervals rather than into complex-shaped continuous sets. Despite being fairly restrictive, our framework allows for non-trivial programs.

Example 1 (Gaussian Mixture Model). The following theory encodes a Gaussian mixture model. The atom `mix(X)` can be used later on as if it were a simple continuous fact, which means the variable `X` can be processed using `above/2`, `below/2` and `ininterval/3`.

```
                 0.6::heads.      tails :- problog_not(heads).
(X,gaussian(0,1))::g(X).          mix(X) :- heads,g(X).
(X,gaussian(5,2))::h(X).          mix(X) :- tails,h(X).
```

The predicate `problog_not/1` is provided by the ProbLog inference engine. It allows one to negate an atom, similar to Prolog's \+, but can only be applied on *ground* probabilistic facts.

The following theory shall be used as running example throughout the paper to define the semantics of Hybrid Problog and to explain the inference algorithm.

Example 2 (Weather). This theory models weather during winter time. The background knowledge states that a person catches a cold when the temperature is below 0° Celsius or when it is colder than 5° Celsius while it rains.

```
                   0.8::rain.     catchcold :- rain,temp(T),below(T,5).
(T,gaussian(2,8))::temp(T).       catchcold :- temp(T),below(T,0).
```

The semantics of Hybrid ProbLog theory $T = F \cup F^c \cup \mathcal{BK}$ is given by probability distributions over subsets of the facts f_i (called *subprograms*), and over sample values for the numeric variables in the continuous facts f_i^c (called *continuous subprograms*). The subprograms $L \subseteq L_F$ are distributed as in ProbLog (cf. Equation (1)), and the continuous subprograms are distributed as described in Section 3.1. Combining both gives one the success probability of queries in a Hybrid ProbLog theory as described in Section 3.2.

3.1 Distribution Over Continuous Subprograms

Let $\Theta^c = \{\theta^c_{j1}, \ldots \theta^c_{ji'_j} | j = 1, \ldots, m\}$ be a finite set of possible substitutions for the non-numeric variables in the continuous facts $(X_j, \phi_j) :: f^c_j$ where i'_j is the number of substitutions for fact j. Each substitution instance $f^c_j \theta^c_{jk}$ is associated with a random variable X_{jk} with probability distribution ϕ_j. The X_{jk} are assumed to be independent. Let $\boldsymbol{X}$ denote the $|\Theta^c|$-dimensional vector of the random variables, and $f(\boldsymbol{x})$ their joint density function. A sample value $\boldsymbol{x}$ for $\boldsymbol{X}$ defines the continuous subprogram $L_{\boldsymbol{x}} := \{f^c_j \theta^c_{jk}\{X_{jk} \leftarrow x_{jk}\} \mid j = 1, \ldots, m; k = 1, \ldots i'_j\}$ where $\{X_{jk} \leftarrow x_{jk}\}$ is the substitution of X_{jk} by x_{jk}.

Example 3 (Continuous Subprogram). Consider the following set of continuous facts where the second fact is *non-ground.* That is, one can obtain several ground instances where each instance has a continuous value drawn independently from the same distribution.

$$(\mathtt{X}, \mathtt{gaussian}(1,2)) :: \mathtt{h}(\mathtt{X}). \qquad (\mathtt{X}, \mathtt{gaussian}(4,3)) :: \mathtt{f}(\mathtt{X}, \mathtt{Y}).$$

When one applies the substitutions $\theta^c_{1,1} = \emptyset$, $\theta^c_{2,1} = \{\mathtt{Y} \leftarrow a\}$, $\theta^c_{2,2} = \{\mathtt{Y} \leftarrow b\}$ together with the point $x_{1,1} = 0.9$, $x_{2,1} = 2.3$, $x_{2,2} = 4.2$, one obtains the *continuous subprogram* $L_{\boldsymbol{x}} = \{\mathtt{h}(0.9), \mathtt{f}(2.3, \mathtt{a}), \mathtt{f}(4.2, \mathtt{b})\}$.

The joint distribution of $\boldsymbol{X}$ thus defines a distribution over continuous subprograms. Specifically, for a $|\Theta^c|$-dimensional interval $I = [a_{\theta^c_{11}}, b_{\theta^c_{11}}] \times \ldots \times [a_{\theta^c_{mi'_m}}, b_{\theta^c_{mi'_m}}]$ (which may also be open or half-open in every dimension), one obtains the probability of the set of continuous subprograms with continuous parameters in I:

$$P_P(\boldsymbol{X} \in I | F^c) := \int_{a_{\theta^c_{11}}}^{b_{\theta^c_{11}}} \cdots \int_{a_{\theta^c_{mi'_m}}}^{b_{\theta^c_{mi'_m}}} f(\boldsymbol{x})\, d\boldsymbol{x} \tag{3}$$

Example 4 (Joint Density). In Example 3, the joint density function is $f(\boldsymbol{x}) = f(x_{1,1}, x_{2,1}, x_{2,2}) = \varphi_{1,2}(x_{1,1}) \times \varphi_{4,3}(x_{1,2}) \times \varphi_{4,3}(x_{2,2})$ where $\varphi_{\mu;\sigma}$ is the density of a normal distribution $\mathcal{N}(\mu, \sigma)$.

3.2 Success Probabilities of Queries

The success probability $P_s(q)$ of a query q is the probability that q is provable in $L \cup L_{\boldsymbol{x}} \cup \mathcal{BK}$, where L is distributed according to (1), and $\boldsymbol{x}$ according to $f(\boldsymbol{x})$ respectively. The key to computing success probabilities is the consideration of admissible intervals, as introduced in the following definition.

Definition 1. *An interval* $I \subseteq \mathbb{R}^{|\Theta^c|}$ *is called* admissible *for a query* q *and a theory* $T = F \cup F^c \cup \mathcal{BK}$ *iff*

$$\forall \boldsymbol{x}, \boldsymbol{y} \in I, \forall L \subseteq L_F : \Big(L \cup L_{\boldsymbol{x}} \cup \mathcal{BK}\Big) \models q \Leftrightarrow \Big(L \cup L_{\boldsymbol{y}} \cup \mathcal{BK}\Big) \models q \tag{4}$$

If (4) *holds, we can also write* $L \cup L_I \cup \mathcal{BK} \models q$.

A partition $\mathcal{A} = I_1, I_2, \ldots, I_k$ *of* $\mathbb{R}^{|\Theta^c|}$ *is called* admissible *for a query* q *and a theory* T *iff all* I_i *are admissible intervals for* q *and* T.

In other words, an admissible interval I is "small enough" such that the values of the continuous variables, as long as they are in I, do not influence the provability of q. Within an admissible interval, the query either always fails or always succeeds for any sampled subset $L \subseteq L_F$ of probabilistic facts.

Example 5 (Admissible Intervals). Consider the Hybrid ProbLog theory consisting out of a single continuous fact:

$$(\texttt{X}, \texttt{gaussian}(1,2)) :: \texttt{h}(\texttt{X})$$

For the query $\texttt{h}(\texttt{X}), \texttt{ininterval}(\texttt{X}, 5, 10)$, the interval $[0, 10]$ is not admissible in this theory. The reason is, that for $x = 4 \in [0, 10]$ the query fails but for $x = 6 \in [0, 10]$ it succeeds. The intervals $[6, 9)$, $[5, 10]$, or $(-\infty, 5)$, for example, are all admissible. Note, that the inference engine allows one to evaluate conjunctive queries and that the predicate `ininterval/3` is automatically added to the background knowledge.

Definition 2 (Discretized Theory). *Let* $T = F \cup F^c \cup \mathcal{BK}$ *be a Hybrid ProbLog theory, then the* discretized theory T_D *is defined as*

$$\begin{aligned} &F \cup \{f^c\{X \leftarrow f^c\} \mid (X, \phi) :: f^c \in F^c\} \\ &\quad \cup \mathcal{BK} \\ &\quad \cup \{\texttt{below}(\texttt{X},\texttt{C}), \texttt{above}(\texttt{X},\texttt{C}), \texttt{ininterval}(\texttt{X},\texttt{C1},\texttt{C2})\} \end{aligned}$$

where $f^c\{X \leftarrow f^c\}$ *is the atom resulting from substituting the variable* X *by the term* f^c.

The substitutions simplify the inference process. Whenever a continuous variable is used in a comparison predicate, the variable will be bound to the original continuous fact. Therefore, one can use a standard proof algorithm without keeping track of continuous variables. The discretized theory still contains probabilistic facts if F is not empty, thus it is a ProbLog theory. Definition 2 allows one to merge the infinite number of proofs, which every Hybrid ProbLog theory has, into a potentially finite number of proofs. This property is needed to compute the admissible intervals efficiently.

Example 6 (Proofs in the discretized theory). The discretized theory T_D for Example 2 is

```
0.8::rain.             catchcold :- rain, temp(T), below(T, 5).
temp(temp(T)).         catchcold:- temp(T), below(T, 0).
below(X, C).           above(X, C).        ininterval(X, C1, C2).
```

The discretized theory contains two proofs for the query `catchcold`. For each proof, one can extract

- f_i the probabilistic facts used in the proof
- c_i the continuous facts used in the proof
- d_i the comparison operators used in the proof

The proofs of `catchcold` can be characterized by:

$$\begin{array}{lll} f_1 = \{\texttt{rain}\} & c_1 = \{\texttt{temp(temp(T))}\} & d_1 = \{\texttt{below(temp(T),5)}\} \\ f_2 = \varnothing & c_2 = \{\texttt{temp(temp(T))}\} & d_2 = \{\texttt{below(temp(T),0)}\} \end{array}$$

It is possible, though not in Example 6, that the same continuous fact is used by several comparison operators within one proof, i.e. $f_i = \{\texttt{below(f(X),10)}, \texttt{above(f(X),0)}\}$. In such cases, one has to build the intersection of all intervals to determine the interval in which all comparison operators succeed, i.e. $f_i = \{\texttt{ininterval(f(X),0,10)}\}$. If that intersection is empty, the proof will fail in the original non-discretized theory. Building the intersections can also be interleaved with proving the goal.

The following theorem guarantees that an admissible partition exists for each query which has finitely many proofs in the discretized theory.

Theorem 1. *For every theory T, every query q that has only finitely many proofs in T_D, and all finite sets of possible substitutions for the probabilistic facts and the continuous facts Θ, Θ^c, there exists a finite partition of $\mathbb{R}^{|\Theta^c|}$that is admissible for T and q.*

Proof. This follows from the fact that conditions defined by `below/2`, `above/2`, and `ininterval/3` are satisfied by intervals of sample values, and finite combinations of such conditions, which may appear in a proof, still define intervals. □

Algorithm 2 can be used to find *admissible partitions*. The algorithm has to be modified as described in Section 4 to respect interval endpoints. Given an admissible partition $\mathcal{A}$ one obtains the success probability of a query q as follows:

$$P_{s,\mathcal{A}}(q|T) := \sum_{L \subseteq L_F} \sum_{\substack{I \in \mathcal{A}: \\ L \cup L_I \cup \mathcal{BK} \models q}} P_P(L|F) \cdot P_P(\boldsymbol{X} \in I|F^c) \tag{5}$$

The following theorem shows that the values of P_s are independent of the partition $\mathcal{A}$ and therefore we can write $P_s(q|T)$ instead of $P_{s,\mathcal{A}}(q|T)$.

Theorem 2. *Let $\mathcal{A}$ and $\mathcal{B}$ be admissible partitions for the query q and the theory T then $P_{s,\mathcal{A}}(q|T) = P_{s,\mathcal{B}}(q|T)$.*

Proof. Proven directly, by showing that for two admissible partitions one can construct a third partition that returns the same success probability. Using the definition for the success probability (5) we get:

$$P_{s,\mathcal{A}}(q|T) := \sum_{L \subseteq L_F} \sum_{\substack{I \in \mathcal{A}: \\ L \cup L_I \cup \mathcal{BK} \models q}} P_P(L|F) \cdot P_P(\boldsymbol{X} \in I|F^c) \tag{6}$$

$$P_{s,\mathcal{B}}(q|T) := \sum_{L \subseteq L_F} \sum_{\substack{I \in \mathcal{B}: \\ L \cup L_I \cup \mathcal{BK} \models q}} P_P(L|F) \cdot P_P(\boldsymbol{X} \in I|F^c) \tag{7}$$

Since $\mathcal{A}$ and $\mathcal{B}$ are both finite, one can construct a partition $\mathcal{C}$ such that it subsumes both $\mathcal{A}$ and $\mathcal{B}$, that is

$$\forall I \in \mathcal{A} : \exists I_1, \ldots, I_n \in \mathcal{C} : I = I_1 \cup \ldots \cup I_n \text{ and}$$
$$\forall I \in \mathcal{B} : \exists I'_1, \ldots, I'_{n'} \in \mathcal{C} : I = I'_1 \cup \ldots \cup I'_{n'}$$

Because $\mathcal{A}$ is admissible and by construction of $\mathcal{C}$, we can represent any summand in (6) as a sum over intervals in $\mathcal{C}$. That is, for each $L \subseteq L_F$ and each $I \in \mathcal{A}$ there exist $I_1, \ldots, I_n \in \mathcal{C}$ such that

$$P_P(L|F) \cdot P_P(\boldsymbol{X} \in I|F^c) = \sum_{i=1}^{n} P_P(L|F) \cdot P_P(\boldsymbol{X} \in I_i|F^c) \ . \tag{8}$$

Because $\mathcal{A}$ is a partition and by construction of $\mathcal{C}$, the intervals needed to cover $I \in \mathcal{A}$ are disjoint from the intervals needed to cover $I' \in \mathcal{A}$ if $I \neq I'$. Therefore

$$\sum_{\substack{I \in \mathcal{A}: \\ L \cup L_I \cup \mathcal{BK} \models q}} P_P(L|F) \cdot P_P(\boldsymbol{X} \in I|F^c) = \sum_{\substack{I \in \mathcal{C}: \\ L \cup L_I \cup \mathcal{BK} \models q}} P_P(L|F) \cdot P_P(\boldsymbol{X} \in I|F^c) \tag{9}$$

for any subprogram $L \subseteq L_F$. From (9) and the definition of the *success probability* (5) follows

$$P_{s,\mathcal{A}}(q|T) = P_{s,\mathcal{C}}(q|T) \ .$$

Similarly, one can show that

$$P_{s,\mathcal{B}}(q|T) = P_{s,\mathcal{C}}(q|T) \ . \qquad \square$$

Theorem 1 and 2 guarantee that the semantics of Hybrid ProbLog, i.e. the fragment that restricts the usage of continuous values, is well-defined. The imposed restrictions provide a balance between expressivity and tractability. They allow one to discretize the space $\mathbb{R}^n$ of possible assignments to the continuous facts in multidimensional intervals such that the actual values within one interval do not matter. In turn, this makes efficient inference algorithms possible. Comparing two continuous values against each other would couple them. This would require a more complicated discretization of $\mathbb{R}^n$ in the form of polyhedra which are harder to represent and to integrate over. Allowing arbitrary functions to be applied on continuous values, eventually, leads to a fragmentation of the space in arbitrary sets. This makes exact inference virtually intractable.

4 Exact Inference

In this section we present an exact inference algorithm for Hybrid ProbLog. Our approach generalizes De Raedt *et al.*'s BDD algorithm [8] and generates a BDD [5] that is evaluated by a slight modification of the original algorithm. The pseudocode is shown in Algorithm 1, 2 and 3. In the remainder of this section, we explain the inference steps on Example 2 and calculate the success probability of the query `catchcold`.

Algorithm 1. The inference algorithm collects all possible proofs and partitions the $\mathbb{R}^n$ space according to the constraints imposed by each proof. The intermediate variables f'_u and c'_u are superfluous and have been added to simplify the explanations in this section.

```
function SuccessProb(query $q$, theory $T$)
    $\{(f_i, c_i, d_i)\}_{1 \leq i \leq m} \leftarrow$ FindAllProofs($T, q$)    ▷ Backtracking in Prolog
    for $c\theta \in \cup_{1 \leq i \leq m} c_i$ do    ▷ Iterate over used ground continuous facts
        $\mathbf{A}_{c\theta} \leftarrow$CreatePartition($\{d_1, \ldots, d_m\}$)
        $\{b_{c\theta,I}\}_{I \in A_{c\theta}} \leftarrow$CreateAUXBodies($\mathbf{A}_{c\theta}$)
    end for
    $u \leftarrow 0$    ▷ # disjoint proofs
    for $i = 1, 2, \ldots, m$ do    ▷ Go over all proofs
        $(\hat{c}_1\hat{\theta}_1, \ldots, \hat{c}_t\hat{\theta}_t) \leftarrow c_i$    ▷ All cont. facts of proof $i$ (this simplifies notation)
        $(\hat{d}_1, \ldots, \hat{d}_t) \leftarrow d_i$    ▷ Intervals in proof $i$ (this simplifies notation)
        for $(I_1, \ldots, I_t) \in \mathbf{A}_{\hat{c}_1\hat{\theta}_1} \times \cdots \times \mathbf{A}_{\hat{c}_t\hat{\theta}_t}$ do    ▷ Go over all possible intervals
            if $(d_1 \cap I_1 \neq \varnothing) \wedge \ldots \wedge (d_t \cap I_t \neq \varnothing)$ then
                $u \leftarrow u + 1$    ▷ Add one more disjoint proof
                $f'_u \leftarrow f_i$    ▷ Probabilistic facts stay the same
                $c'_u \leftarrow c_i$    ▷ Continuous facts stay the same
                $d'_u \leftarrow \{\text{dom}(\hat{c}_1\hat{\theta}_1) = I_1, \ldots, \text{dom}(\hat{c}_t\hat{\theta}_t) = I_t\}$    ▷ Domains are adapted
                $f''_u \leftarrow f_i \cup \{b_{c\theta,I} | c\theta \in c'_u, I \in d'_u\}$    ▷ Add aux. bodies to the facts
            end if
        end for
    end for
    $BDD \leftarrow$GenerateBDD($\bigvee_{1 \leq i \leq u} \bigwedge_{f \in f''_i} f$)    ▷ cf. [9]
    return Prob(root($BDD$))    ▷ cf. [8]
end function
```

1. All proofs for `catchcold` are collected by SLD resolution (Line 2 in Algorithm 1).

$$f_1 = \{\texttt{rain}\} \qquad c_1 = \{\texttt{temp(T)}\} \qquad d_1 = \{\texttt{T} \in (-\infty, 5)\}$$
$$f_2 = \varnothing \qquad c_2 = \{\texttt{temp(T)}\} \qquad d_2 = \{\texttt{T} \in (-\infty, 0)\}$$

   Each proof is described by a set of probabilistic facts f_i, a set of continuous facts c_i, and an interval for each continuous variable in c_i. When a continuous fact is used within a proof, it is added to c_i and the corresponding variable X is added to d_i with $X \in (-\infty, \infty)$.
   When later on `above`(X,c_1) is used in the same proof, the interval I stored in d_i is replaced by $I \cap (c_1, \infty)$, similarly for `below`(X,c_2) it is replaced by $I \cap (-\infty, c_2)$, and for `ininterval`(X, c$_1$, c$_2$) it is replaced by $I \cap [c_1, c_2]$,
2. We partition $\mathbb{R}^1$ because one continuous fact is used. The loop in Line 3 of Algorithm 1 iterates over $(\cup_{1 \leq i \leq m} c_i) = \{\texttt{temp(T)}\}$. When calling the function CreatePartition($\{d_1, d_2\}$) (cf. Algorithm 2) we obtain the admissible partition $\{(-\infty, 0), [0, 5), [5, \infty)\}$ which is used to disjoin the proofs with respect to the continuous facts (Line 8-20 in Algorithm 1):

Algorithm 2. This function returns a partition of $\mathbb{R}$ by creating intervals touching the given intervals. Partitions of $\mathbb{R}^n$ can be obtain by building the cartesian product over the partitions for each dimension. This is possible due to the restrictions imposed in Section 3.

```
 1: function CreatePartition(Set of intervals $D = \{d_1, \dots d_m\}$)
 2:     $C \leftarrow \bigcup_{i=1}^{m} \{low_i, high_i\}$        ▷ $low_i$ and $high_i$ are interval endpoints of $d_i$
 3:     $C \leftarrow C \cup \{-\infty, \infty\}$        ▷ add upper and lower limit of $\mathbb{R}$
 4:     $(c'_1, \dots, c'_k) \leftarrow$SortAndIgnoreDuplicates($c$)
 5:     Result $\leftarrow \{ \ (-\infty, c'_2], (c'_{k-1}, \infty) \ \}$        ▷ $c'_1 = -\infty$ and $c'_k = \infty$
 6:     for $i = 2, \dots, k-1$ do
 7:         Result $\leftarrow$ Result $\cup \{ \ (c'_{i-1}, c'_i] \ \}$
 8:     end for
 9:     return Result
10: end function
```

$$f'_1 = \{\texttt{rain}\} \qquad c'_1 = \{\texttt{temp(T)}\} \qquad d'_1 = \{\texttt{T} \in (-\infty, 0)\}$$
$$f'_2 = \{\texttt{rain}\} \qquad c'_2 = \{\texttt{temp(T)}\} \qquad d'_2 = \{\texttt{T} \in [0, 5)\}$$
$$f'_3 = \varnothing \qquad c'_3 = \{\texttt{temp(T)}\} \qquad d'_3 = \{\texttt{T} \in (-\infty, 0)\}$$

3. We create one auxiliary fact per continuous fact and interval. They are dependent, i.e. $\texttt{temp(T)}_{[-\infty,0)}$ and $\texttt{temp(T)}_{[0,5)}$ cannot be true at the same time. We have to make the dependencies explicit by adding, conceptually, the following clauses to the theory and replacing calls to continuous facts and background predicates by the bodies $b_{c\theta,I}$:

$$\texttt{call\_temp(T)}_{(-\infty,0)} \text{ :- } \texttt{temp(T)}_{(-\infty,0)}$$
$$\texttt{call\_temp(T)}_{[0,5)} \text{ :- } \neg\texttt{temp(T)}_{(-\infty,0)}, \texttt{temp(T)}_{[0,5)}$$
$$\texttt{call\_temp(T)}_{[5,\infty)} \text{ :- } \neg\texttt{temp(T)}_{(-\infty,0)}, \neg\texttt{temp(T)}_{[0,5)}, \texttt{temp(T)}_{[5,\infty)}$$

Only one of the clauses can be true at the same time. The bodies encode a linear chain of decisions. The probability attached to an auxiliary fact $\texttt{temp(T)}_{[l,h)}$ is the conditional probability that the sampled value of T is in the interval $[l, h)$ given it is not in $(-\infty, l)$

$$P\left(\texttt{temp(T)}_{[l,h]}\right) := \left[\int_{l}^{h} \mathcal{N}(2, 8, x)\, dx\right] \cdot \left[1 - \int_{-\infty}^{l} \mathcal{N}(2, 8, x)\, dx\right]^{-1} \qquad (10)$$

where $\mathcal{N}$ is the density of the Gaussian specified for temp/1 in the program. This encodes a switch (cf. [11]) such that the success probability of $\texttt{call\_temp(T)}_{[l,h]}$ is exactly $\int_l^h \mathcal{N}(2, 8, x)\, dx$. To evaluate the cumulative density function, we use the function Phi as described in [10]. If we want to use any other distribution, we have to only replace the evaluation function of the

Algorithm 3. To evaluate the BDD we run a modified version of De Raedt *et al.*'s algorithm that takes the conditional probabilities for each continuous node into account. For Gaussian-distributed continuous facts we use the function Phi [10] to evaluate $\int_{l_i}^{h_i} \mathcal{N}(x, \mu_i, \sigma_i)\, dx$ which performs a Taylor approximation of the cumulative density function (CDF). If the program uses distributions other than Gaussians, the user has to provide the corresponding CDF.

```
1: function Prob(node $n$)
2:     if $n$ is the 1-terminal then return 1
3:     if $n$ is the 0-terminal then return 0
4:     Let $h$ and $l$ be the high and low children of $n$
5:     $p_h \leftarrow$ Prob($h$)
6:     $p_l \leftarrow$ Prob($l$)
7:     if $n$ is a continuous node with attached interval $[a,b]$ and density $\phi_n$ then
8:         $p \leftarrow \left[\int_{l_n}^{h_n} \phi_n(x)\, dx\right] \cdot \left[1 - \int_{-\infty}^{l_n} \phi_n(x)\, dx\right]^{-1}$
9:     else
10:        $p \leftarrow p_n$        ▷ the probability attached to the fact in the ProbLog program
11:    end if
12:    return $p \cdot p_h + (1-p) \cdot p_l$
13: end function
```

density, as all the rest of the algorithm does not depend on the particular distribution. Adding the bodies of the auxiliary clauses to f_i' yields the final set of proofs (Line 17 in Algorithm 1):

$$f_1'' = \{\texttt{rain}, \texttt{temp(T)}_{(-\infty,0)}\}$$
$$f_2'' = \{\texttt{rain}, \neg\texttt{temp(T)}_{(-\infty,0)}, \texttt{temp(T)}_{[0,5)}\}$$
$$f_3'' = \{\texttt{temp(T)}_{(-\infty,0)}\}$$

The proofs are now disjoint with respect to the continuous facts. That is, either the intervals for continuous facts are disjoint or identical. With respect to the probabilistic facts, they are not disjoint and summing up the probabilities of all proofs would yield a wrong result. One would count the probability mass of the overlapping parts multiple times [8].

4. To account for that, we translate the proofs into a Boolean expression in disjunctive normal form (cf. Line 21 in Algorithm 1) and represent it as BDD (cf. Figure 1). This step is similar to ProbLog's inference mechanism

$$\begin{aligned}&\left(\texttt{rain} \wedge \texttt{temp(T)}_{(-\infty,0)}\right)\\ \vee&\left(\texttt{rain} \wedge \neg\texttt{temp(T)}_{(-\infty,0)} \wedge \texttt{temp(T)}_{[0,5)}\right)\\ \vee&\left(\texttt{temp(T)}_{(-\infty,0)}\right)\end{aligned}$$

5. We evaluate the BDD with Algorithm 3 and get the success probability of `catchcold`. This is a slight modification of De Raedt *et al.*'s algorithm (cf. [8,9] for the details) that takes into account the continuous nodes.

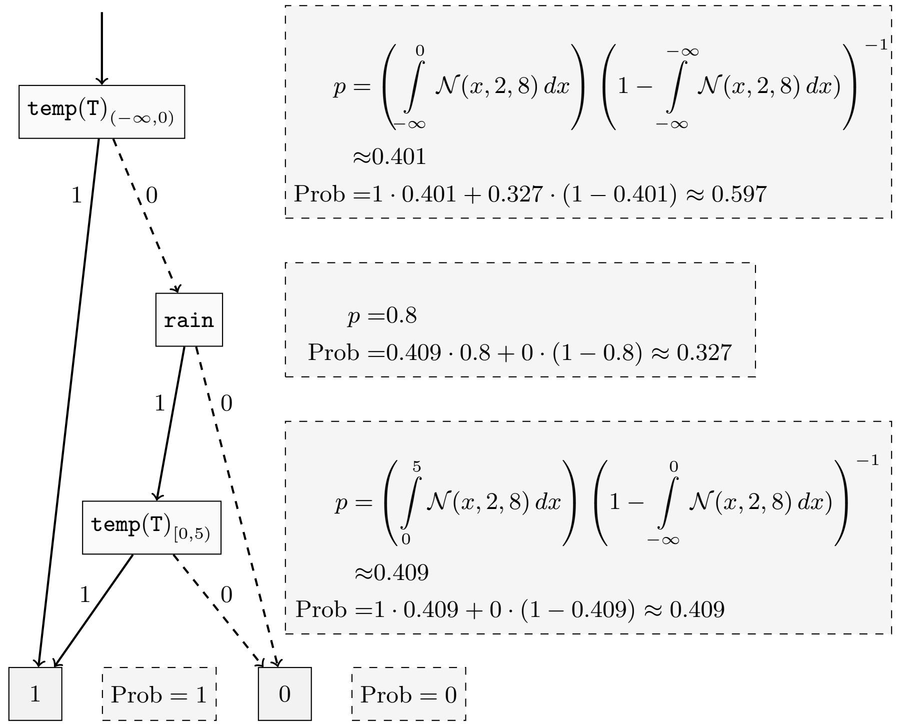

temp(T)$_{(-\infty,0)}$
1
0
rain
1
0
temp(T)$_{[0,5)}$
1
0
1
Prob = 1
0
Prob = 0
$p = \left(\int_{-\infty}^{0} \mathcal{N}(x,2,8)\,dx\right)\left(1-\int_{-\infty}^{-\infty}\mathcal{N}(x,2,8)\,dx\right)^{-1} \approx 0.401$
Prob $=1\cdot 0.401 + 0.327\cdot(1-0.401) \approx 0.597$
$p = 0.8$
Prob $=0.409\cdot 0.8 + 0\cdot(1-0.8) \approx 0.327$
$p = \left(\int_{0}^{5} \mathcal{N}(x,2,8)\,dx\right)\left(1-\int_{-\infty}^{0}\mathcal{N}(x,2,8)\,dx\right)^{-1} \approx 0.409$
Prob $=1\cdot 0.409 + 0\cdot(1-0.409) \approx 0.409$

Fig. 1. This BDD [5] encodes all proofs of `catchcold` in the theory from Example 2. The dashed boxes show the intermediate results while traversing the BDD with Algorithm 3. The success probability of the query is returned at the root and is 0.597.

The function CreatePartition (cf. Algorithm 2) does not necessarily return an *admissible partition* as it ignores the interval endpoints by creating right-open intervals. For instance, if one obtains two proofs which impose as constraint the intervals $[1, 2]$ and $[2, 3)$, the minimal admissible partition is

$$\{(-\infty, 1), [1, 2), [2, 2], (2, 3), [3, \infty)\}\ .$$

The function CreatePartition, however, returns the inadmissible partition

$$\{(-\infty, 1), [1, 2), [2, 3), [3, \infty)\}\ .$$

With respect to the interval $[2, 3)$, the first proof succeeds for $x = 2$ but fails for any other value in $[2, 3)$ – which is not allowed for admissible intervals (cf. Definition 1). Though, when calculating the *success probability*, one can ignore interval endpoints. Since the calculation involves integrals of the form $\int_{l_i}^{h_i} \phi(x)\,dx$, it is irrelevant whether intervals are open or closed (cf. Equation (10)). Also, an integral over a single point interval has value 0.

However, when one wants to know whether there is a proof with specific values for some or all continuous facts, one has to be precise about the interval endpoints

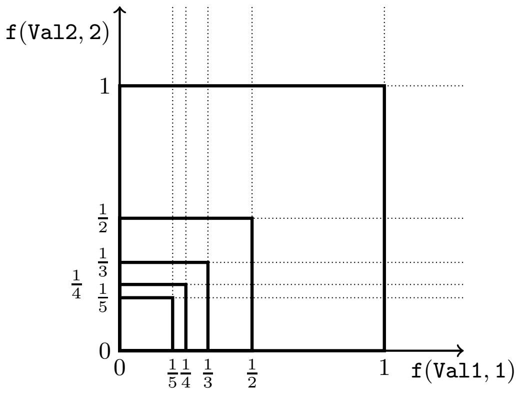


Fig. 2. The query $\mathtt{s(5,2)}$ succeeds, if both values $\mathtt{f(Val1,1)}$ and $\mathtt{f(Val2,2)}$ lie in one of the intervals $[0, \frac{1}{1}]$, $[0, \frac{1}{2}]$, ..., $[0, \frac{1}{5}]$. These areas correspond to the thick-lined squares starting at $(0, 0)$. Since they overlap, one has to partition the space $\mathbb{R}^2$ in order to disjoin the proofs. Due to the restrictions of Hybrid ProbLog program, i.e., continuous variables cannot be compared against each other, one can obtain an admissible partition for each dimension independently. Algorithm 2 returns the partitions shown by the dotted lines. The horizontal lines partition the space of $\mathtt{f(Val2,2)}$ and the vertical the space of $\mathtt{f(Val1,1)}$.

and use a modified algorithm. Admissibility can be ensured, for instance, by creating for each pair of constants the open interval (l_i, h_i) and the single point interval $[l_i, h_i]$. For the former example this is

$$\{(-\infty, 1), [1, 1], (1, 2), [2, 2], (2, 3), [3, 3], (3, \infty)\} \ .$$

5 Experiments

We implemented Hybrid ProbLog in YAP 6.0.7 and set up experiments to answer the question

> How does the inference algorithm (cf. Algorithm 1) scale in the size of the partitions and in the number of ground continuous facts?

In domains where exact inference is feasible, the number of continuous facts and comparison operations is typically small compared to the rest of the theory. Our algorithm is an intermediate step between SLD resolution and BDD generation. Therefore, it is useful to know how much the disjoining operations cost compared to the other inference steps. We tested our algorithm on the following theory:

```
(Val, gaussian(0, 1)) :: f(Val, ID).
    s(Consts, Facts) :- between(1, Facts, ID), between(1, Consts, Top),
                        High is Top/Consts,
                        f(Val, ID), ininterval(Val, 0, High).
```

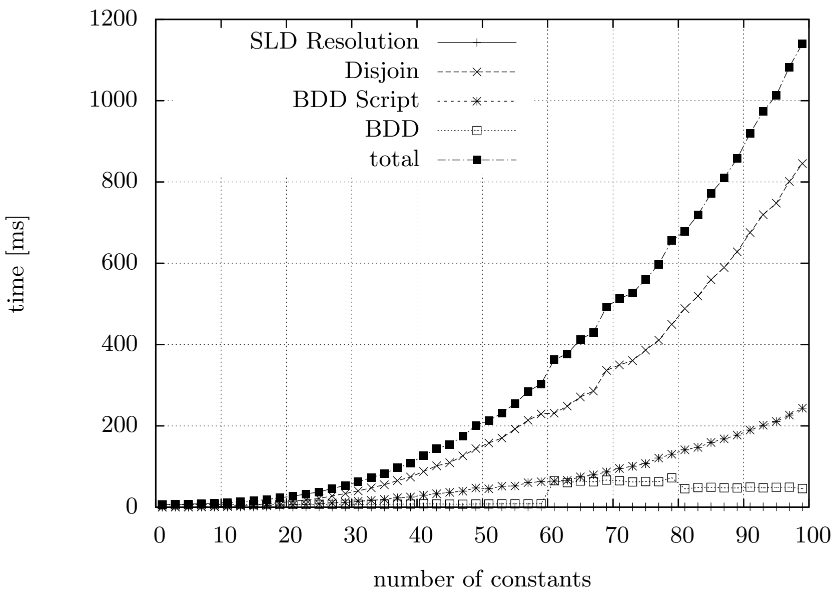


Fig. 3. Runtimes for calculating the success probability of $\texttt{s}(\texttt{Consts}, \texttt{Facts})$ for varying the number of constants $s(1, 1), \cdots s(100, 1)$. As the graphs show, most of the time is spent on disjoining the proofs, that is partitioning the domains. The BDD time stays more or less constant, this is due to the fact that the resulting Boolean expression, i.e. $\neg\texttt{s}(\texttt{Val}, 1)_{(-\infty,0)} \wedge \left(\texttt{s}(\texttt{Val}, 1)_{[0,\frac{1}{n})} \vee \texttt{s}(\texttt{Val}, 1)_{[\frac{1}{n},\frac{1}{n-1})} \vee \ldots \vee \texttt{s}(\texttt{Val}, 1)_{[\frac{1}{2},1)}\right)$, is rather simple. This can be detected and exploited by the BDD package.

The query $\texttt{s}(\texttt{Consts}, \texttt{Facts})$ has $\texttt{Consts} \times \texttt{Facts}$ many proofs where both arguments have to be positive integers. The first argument determines the number of partitions needed to disjoin the proofs with respect to the continuous facts and the second argument determines how many continuous facts are used. The query $\texttt{s}(5, 2)$, for instance, uses two continuous facts, $\texttt{f}(\texttt{Val1}, 1)$ and $\texttt{f}(\texttt{Val2}, 1)$, and compares them to the intervals $[0, 1], [0, \frac{1}{2}], \cdots [0, \frac{1}{5}]$. Figure 2 shows the resulting partitioning when proving the query $\texttt{s}(5, 2)$. In general, one obtains $(\texttt{Consts} + 1)^{\texttt{Facts}}$ many partitions of the space $\mathbb{R}^{\texttt{Facts}}$.

The success probability of $\texttt{s}(\texttt{Consts}, \texttt{Facts})$ is independent of $\texttt{Consts}$. That is, for fixed $\texttt{Facts}$ and any $c_1, c_2 \in \mathbb{N}$

$$P_s(\texttt{s}(c_1, \texttt{Facts})|T) = P_s(\texttt{s}(c_2, \texttt{Facts})|T)$$

We ran two series of queries[3]. First, we used one continuous fact and varied the number of constants from 1 to 100. As the graph in Figure 3 shows, the disjoin

[3] The experiments were performed on an Intel Core 2 Quad machine with 2.83GHz and 8GB of memory. We used the CUDD package for BDD operations and set the reordering heuristics to CUDD_REORDER_GROUP_SIFT. Each query has been evaluated 20 times and the runtimes were averaged.

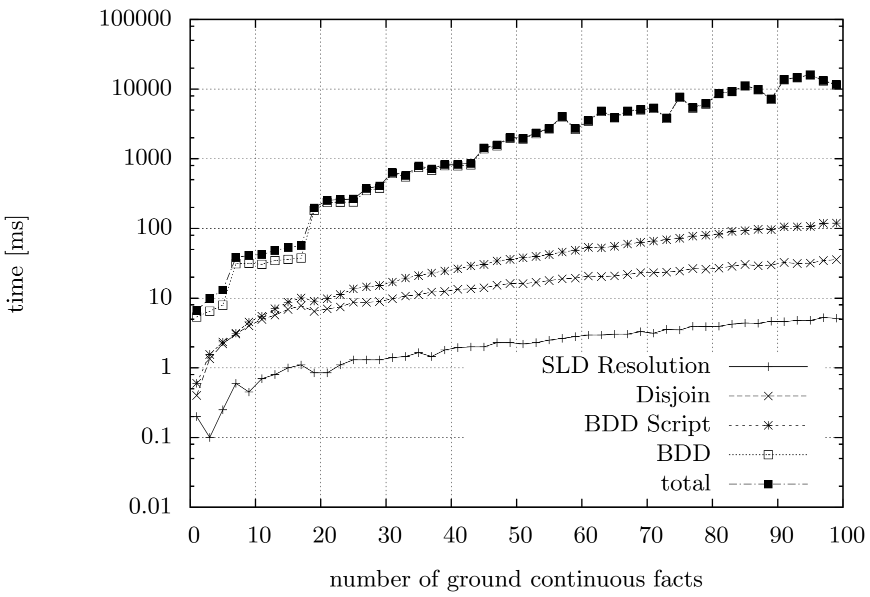


Fig. 4. Runtimes for calculating the success probability of $\mathtt{s}(\mathtt{Consts}, \mathtt{Facts})$ for varying the number of dimensions $s(5, 1), \ldots, s(5, 100)$. In this setting, most of the time is spent in the BDD package, that is building the BDD based on the script and traversing it. The runtime for our disjoin operation grows only linearly. This is due to the fact that the partitions of $\mathbb{R}^n$ can be factorized into each dimensions because we do not allow comparisons between two continuous variables.

operation – that is finding all partitions, generating the auxiliary bodies and rewriting the proofs – runs in $O(\mathtt{Consts}^2)$ when everything else stays constant. In this case, due to compression and pruning operations during the BDD script generation [9] (the input for the BDD package), building and evaluating the BDD runs in quasi-linear time. In the second run, we varied the number of continuous facts by evaluating the queries $\mathtt{s}(5, 1), \cdots, \mathtt{s}(5, 100)$. As Figure 4 shows, our algorithm (depicted by the Disjoin graph) runs in linear time. The runtime for the BDD operations grows exponentially due to the reordering heuristics used by the BDD package.

6 Conclusions and Future Work

We extended ProbLog with continuous distributions and introduced an exact inference algorithm. The expressivity has been restricted to make inference tractable. Possible directions for future work include comparisons between two continuous facts and applying functions on continuous values. We are working on upgrading existing ProbLog parameter learning methods to continuous facts. ProbLog is available for download at `http://dtai.cs.kuleuven.be/problog`.

Acknowledgments

Bernd Gutmann is supported by the Research Foundation-Flanders (FWO-Vlaanderen). This work is supported by the GOA/08/008 project "Probabilistic Logic Learning".

References

1. Vennekens, J., Denecker, M., Bruynooghe, M.: Representing causal information about a probabilistic process. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, pp. 452–464. Springer, Heidelberg (2006)
2. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Sterling, L. (ed.) Proceedings of the Twelfth International Conference on Logic Programming (ICLP 1995), pp. 715–729. MIT Press, Cambridge (1995)
3. Milch, B., Marthi, B., Russell, S.J., Sontag, D., Ong, D.L., Kolobov, A.: Blog: Probabilistic models with unknown objects. In: Kaelbling, L.P., Saffiotti, A. (eds.) IJCAI 2005: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, pp. 1352–1359. Professional Book Center (2005)
4. Goodman, N., Mansinghka, V.K., Roy, D.M., Bonawitz, K., Tenenbaum, J.B.: Church: a language for generative models. In: McAllester, D.A., Myllymäki, P. (eds.) UAI 2008: Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, July 9-12, pp. 220–229. AUAI Press, Helsinki (2008)
5. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers 35(8), 677–691 (1986)
6. Wang, J., Domingos, P.: Hybrid markov logic networks. In: Fox, D., Gomes, C.P. (eds.) Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, July 13-17, pp. 1106–1111. AAAI Press, Menlo Park (2008)
7. Murphy, K.: Inference and learning in hybrid bayesian networks. Technical Report UCB/CSD-98-990, University of Berkeley (1998)
8. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discovery. In: Veloso, M.M. (ed.) IJCAI 2007: Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, pp. 2462–2467 (2007)
9. Kimmig, A., Santos Costa, V., Rocha, R., Demoen, B., De Raedt, L.: On the efficient execution of probLog programs. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 175–189. Springer, Heidelberg (2008)
10. Marsaglia, G.: Evaluating the normal distribution. Journal of Statistical Software 11(5), 1–11 (2004)
11. De Raedt, L., Demoen, B., Fierens, D., Gutmann, B., Janssens, G., Kimmig, A., Landwehr, N., Mantadelis, T., Meert, W., Rocha, R., Santos Costa, V., Thon, I., Vennekens, J.: Towards digesting the alphabet-soup of statistical relational learning. In: Roy, D., Winn, J., McAllester, D., Mansinghka, V., Tenenbaum, J. (eds.) Proceedings of the 1st Workshop on Probabilistic Programming: Universal Languages, Systems and Applications, Whistler, Canada (December 2008)

Multivariate Prediction for Learning on the Semantic Web

Yi Huang[1], Volker Tresp[1], Markus Bundschus[2], Achim Rettinger[3], and Hans-Peter Kriegel[2]

[1] Siemens AG, Corporate Technology, Munich, Germany
[2] Ludwig-Maximilians-Universität München, Munich, Germany
[3] Karlsruhe Institute of Technology, Karlsruhe, Germany

Abstract. One of the main characteristics of Semantic Web (SW) data is that it is notoriously incomplete: in the same domain a great deal might be known for some entities and almost nothing might be known for others. A popular example is the well known friend-of-a-friend data set where some members document exhaustive private and social information whereas, for privacy concerns and other reasons, almost nothing is known for other members. Although deductive reasoning can be used to complement factual knowledge based on the ontological background, still a tremendous number of potentially true statements remain to be uncovered. The paper is focused on the prediction of potential relationships and attributes by exploiting regularities in the data using statistical relational learning algorithms. We argue that multivariate prediction approaches are most suitable for dealing with the resulting high-dimensional sparse data matrix. Within the statistical framework, the approach scales up to large domains and is able to deal with highly sparse relationship data. A major goal of the presented work is to formulate an inductive learning approach that can be used by people with little machine learning background. We present experimental results using a friend-of-a-friend data set.

1 Introduction

The Semantic Web (SW) is becoming a reality. Most notably is the development around the Linked Open Data (LOD) initiative, where the term Linked Data is used to describe a method of exposing, sharing, and connecting data via dereferenceable Unique Resource Identifiers (URIs) on the Web. Typically, existing data sources are published in the Semantic Web's Resource Description Framework (RDF), where statements are expressed as simple subject-property-object (s, p, o) triples and are graphically displayed as a directed labeled link between a node representing the subject and a node representing the object (Figure 1). Data sources are interlinked with other data sources in the LOD cloud. In some efforts, subsets of the LOD cloud are retrieved in repositories and some form of logical reasoning is applied to materialize implicit triples. The number of inferred triples is typically on the order of the number of explicit triples. One can certainly assume that there are a huge number of additional true triples which are neither known as facts nor can be derived from reasoning. This might concern both triples within one of the contributing data sources such as DBpedia[1] (intralinks), and triples describing interlinks between the contributing data sources. The goal of the work presented here is to estimate the truth values of triples exploiting patterns in the data. Here we need to take into account the nature of the SW. LOD data is currently dynamically evolving and quite noisy. Thus flexibility and ease of use are preferred properties if compared to highly sophisticated approaches that can only be applied by machine learning experts. Reasonable requirements are as follows:

- Machine learning should be “push-button” requiring a minimum of user intervention.
- The learning algorithm should scale well with the size of the SW.
- The triples and their probabilities, which are predicted using machine learning, should easily be integrated into SPARQL-type querying.[2]
- Machine learning should be suitable to the data situation on the SW with sparse data (e.g., only a small number of persons are friends) and missing information (e.g., some people don’t reveal private information).

Looking at the data situation, there are typically many possible triples associated with an entity (these triples are sometimes called entity molecules or, in our work, statistical unit node set) of which only a small part is known to be true. Due to this large degree of sparsity of the relationship data in the SW, multivariate prediction is appropriate for SW learning. The rows, i.e., data points in the learning matrix are defined by the key entities or statistical units in the sample. The columns are formed by nodes that represent the truth values of triples that involve the statistical units. Nodes representing aggregated information form the inputs. The size of the training data set is under the control of the user by means of sampling. Thereby the data matrix size, and thus also training time, can be made independent or only weakly dependent on the overall size of the SW. For the experiments in this paper we use the friend-of-a-friend (FOAF) data set, which is a distributed social domain describing persons and their relationships in SW-format. Our approach is embedded in a statistical framework requiring the definition of a statistical unit and a population. In our experiments we compare different sampling approaches and analyze generalization on a test set.

The paper is organized as follows. In the next section we discuss related work, In Section 3 we discuss how machine learning can be applied to derive probabilistic weights for triples whose truth values are unknown and introduce our approach. In Section 4 we present experimental results using a friend-of-a-friend (FOAF) data set. Finally, Section 5 contains conclusions and outlines further work.

2 Related Work

The work on inductive databases [1] pursues similar goals but is focussed on the less-problematic data situation in relational databases. In [2] the authors describe SPARQL-ML, a framework for adding data mining support to SPARQL. SPARQL-ML was

[1] http://dbpedia.org/

[2] SPARQL is a new standard for querying RDF-specific information and for displaying querying results.

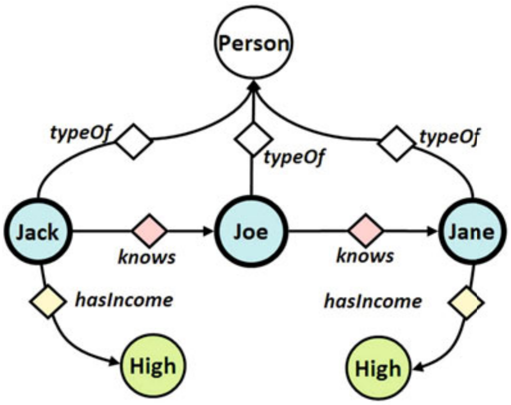


Fig. 1. Example of an RDF graph displaying a social friendship network in which the income of a person is an attribute. Resources are represented by circular nodes and triples are represented by labeled directed links from subject node to object node. The diamond-shaped nodes stand for random variables which are in state *one* if the corresponding triples exist. Nodes representing statistical units (here: *Persons*) have a darker rim.

inspired by Microsoft's Data Mining Extension (DMX). A particular ontology for specifying the machine learning experiment is developed. The SRL methods in [2] are ILP-type approaches based on a closed-world assumption (Relational Bayes Classifier (RBC) and Relational Probabilistic Trees (RPT)). This is in difference to the work presented here, which maintains more of an open-world assumption that is more appropriate in the context of the SW. Another difference is that in our work, both model training and statement prediction can be performed off-line, if desired. In this case inferred triples with their associated certainty values can be stored , e.g., in a triple store, enabling fast query execution.

Unsupervised approaches (examples that are suitable for the relational SW domain are [3–6]) are quite flexible and interpretable and provide a probability distribution over a relational domain. Although unsupervised approaches are quite attractive, we fear that the sheer size of the SW and the huge number of potentially true statements make these approaches inappropriate for Web-scale applications. Supervised learning, where a model is trained to make a prediction concerning a single random variable typically shows better predictive performance and better scalability. Typical examples are many ILP approaches [7, 8] and propositionalized ILP approaches [9, 10]. Multivariate prediction generalizes supervised learning to predict several variables jointly, conditioned on some inputs. The improved predictive performance in multivariate prediction, if compared to simple supervised learning, has been attributed to the sharing of statistical strength between the multiple tasks, i.e., data is used more efficiently see [11] and citations therein for a review). Due to the large degree of sparsity of the relationship data in the SW, we expect that multivariate prediction is quite interesting for SW learning and we will apply it in the following.

3 Statistical Modeling

3.1 Defining the Sample

We must be careful in defining the statistical unit, the population, the sampling procedure and the features. A statistical unit is an object of a certain type, e.g., a person. The population is the set of statistical units under consideration. In our framework, a population might be defined as the set of persons that attend a particular university. For learning we use a subset of the population. In the experimental section we will explore various sampling strategies. Based on the sample, a data matrix is generated where the statistical units in the sample define the rows.

3.2 The Random Variables in the Data Matrix

We now introduce for each potential triple a *triple node* drawn as a diamond-shaped node in Figure 1. A triple node is in state *one* (*true*) if the triple is known to exist and is in state *zero* (*false*) if the triple is known not to exist. Graphically, one only draws the triple nodes in state *one*, i.e., the existing triples.

We now associate some triples with statistical units. The idea is to assign a triple to a statistical unit if the statistical unit appears in the triple. Let's consider the statistical unit *Jane*. Based on the triples she is participating in, we obtain *(?personA, typeOf, Person)*, *(Joe, knows, ?personA)*, and *(?personA, hasIncome, High)* where *?personA* is a variable that represents a statistical unit. The expressions form the random variables (outputs) and define columns in the data matrix.[3] By considering the remaining statistical units *Jack* and *Joe* we generate the expressions (columns), *(?personA, knows, Jane)* and *(Jack, knows, ?personA)*. We will not add *(Jane, knows, ?personA)* since Jane considers no one in the data base to be her friend. We iterate this procedure for all statistical units in the sample and add new expressions (i.e., columns in the data matrix), if necessary. Note that expressions that are not represented in the sample will not be considered. Also, expressions that are rarely true (i.e., for few statistical units) will be removed since no meaningful statistics can be derived from few occurrences. In [12] the triples associated with a statistical unit were denoted as *statistical unit node set* (SUNS). The matrix formed with the N statistical units as rows and the random variables as columns is denoted as Y.

3.3 Non-random Covariates in the Data Matrix

The columns we have derived so far represent truth values of actual or potential triples. Those triples are treated as random variables in the analysis. If the machine learning algorithm predicts that a triple is very likely, we can enter this triple in the data store. We now add columns that provide additional information for the learning algorithm but which we treat as covariates or fixed inputs.

First, we derive simplified relations from the data store. More precisely, we consider the expressions derived in the last subsection and replace constants by variables. For

[3] Don't confuse a random variable representing the truth value of a statement with a variable in a triple, representing an object.

example, from *(?personA, knows, Jane)* we derive *(?personA, knows, ?personB)* and count how often this expression is true for a statistical unit *?personA*, i.e., we count the number of friends of person *?personA*.

Second, we consider a simple type of aggregated features from outside a SUNS. Consider first a binary triple *(?personA, knows, Jane)* . If Jane is part of another binary triple, in the example, *(?personA, hasIncome, High)* then we form the expression *(?personA, knows, ?personB)* $\wedge$ *(?personB, hasIncome, High)* and count how many rich friends a person has. A large number of additional features are possible but so far we restricted ourselves to these two types. The matrix formed with the N statistical units as rows and the additional features as columns is denoted as X.

After construction of the data matrix we prune away columns in X and in Y which have *ones* in fewer than ϵ percent of all rows, where ϵ is usually a very small number. This is because for those features no meaningful statistical analysis is possible. Note that by applying this pruning procedure we reduce the exponential number of random variables to typically a much smaller set.

3.4 Algorithms for Learning with Statistical Units Node Sets

In a statistical setting as described above, the statistical unit node set (SUNS) is defined mostly based on local neighborhoods of statistical units. By adding aggregated information derived from the neighborhood, homophily can also be modeled. For instance, the income of a person can be predicted by the average income of this person's friends.

As we will see in the experiments, the resulting data matrices are typically high-dimensional and sparse. In this situation, multivariate prediction approaches have been most successful [11]. In multivariate prediction all outputs are jointly predicted such that statistical strength can be shared between outputs. The reason is that some or all model parameters are sensitive to all outputs, improving the estimates of those parameters.[4]

We apply four different multivariate prediction approaches. First, we utilize a reduced rank penalized regression (RRPP) algorithm to obtain an estimated matrix via the formula

$$\hat{Y} = U_r \, \text{diag}_r \left(\frac{d_k}{d_k + \lambda} \right) U_r^T Y$$

where d_k and U_r are derived from a r-rank eigen decomposition of the kernel matrix $K \approx U_r D_r U_r^T$. U_r is a $N \times r$ matrix with r orthonormal columns, $\text{diag}_r \left(\frac{d_k}{d_k+\lambda} \right)$ is a diagonal matrix containing the r largest eigen values and λ is a regularization parameter. The kernel matrix K can be defined application specifically. Typically, as in the following application, one works with a linear kernel defined by $K = ZZ^T$, where $Z = [\alpha X, Y]$ is formed by concatenating X and Y and where α is a positive weighting factor.[5]

[4] Although the completion is applied to the entire matrix, only *zeros* —representing triples with unknown truth values— are overwritten.

[5] Alternatively, we can define a linear kernel solely based on the input attributes $K = XX^T$, when $\alpha \rightarrow \infty$, or solely based on the output attributes $K = YY^T$, when $\alpha = 0$.

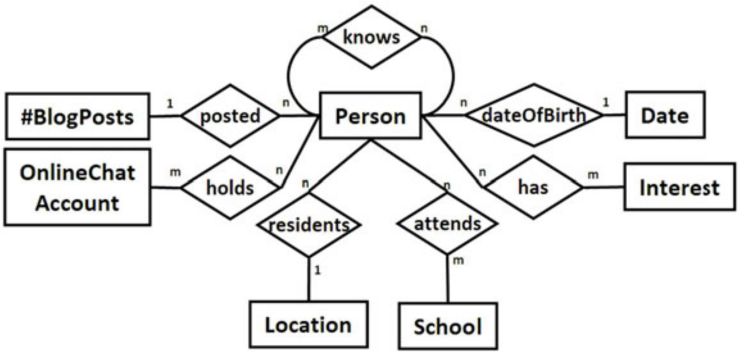


Fig. 2. Entity-relationship diagram of the LJ-FOAF domain

Besides RRPP we investigate three other multivariate prediction approaches based on matrix completion, i.e., singular value decomposition (SVD), non-negative matrix factorization (NNMF) [13] and latent Dirichlet allocation (LDA) [14]. All approaches estimate unknown matrix entries via a low-rank matrix approximation. NNMF is a decomposition under the constraints that all terms in the factoring matrices are non-negative. LDA is based on a Bayesian treatment of a generative topic model. After matrix completion of the *zero* entries in the data matrix, the entries are interpreted as certainty values that the corresponding triples are true. After training, the models can also be applied to statistical units in the population outside the sample.

4 Experiments

4.1 Data Set and Experimental Setup

Data Set. The experiments are based on friend-of-a-friend (FOAF) data. The purpose of the FOAF project [15] is to create a web of machine-readable pages describing people, their relationships, and people's activities and interests, using W3C's RDF technology. The FOAF ontology is based on RDFS/OWL and is formally specified in the FOAF Vocabulary Specification 0.91[6].

We gathered our FOAF data set from user profiles of the community website LiveJournal.com[7]. All extracted entities and relations are shown in Figure 2. In total we collected 32,062 persons and all related attributes. An initial pruning step removed little connected persons and rare attributes. Table 1 lists the number of different individuals (top rows) and their known instantiated relations (bottom rows) in the full triple set, in the pruned triple set and in triples sets in different experiment settings (explained below). The resulting data matrix, after pruning, has 14,425 rows (persons) and 15,206 columns. Among those columns 14,425 ones (friendship attributes) refer to the property *knows*. The remaining 781 columns (general attributes) refer to general information about age, location, number of blog posts, attended school, online chat account and interest.

[6] http://xmlns.com/foaf/spec/

[7] http://www.livejournal.com/bots/

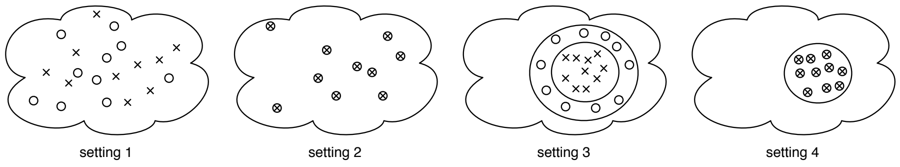


Fig. 3. Evaluated sampling strategies

Data Retrieval and Sampling Strategies. In our experiments we evaluated the generalization capabilities of the learning algorithms given eight different settings. The first four settings are illustrated in Figure 3. A cloud symbolizes the part of the Web that can effectively be accessed (in our case the data set given in Table 1). Crosses represent persons that are known during the training phase (training set) and circles represent persons with *knows* relations that need to be predicted.

Setting 1 describes the situation where the depicted part of the SW is randomly accessible, meaning that all instances can be queried directly from triple stores. Statistical units in the sample for training are randomly sampled and statements for other randomly selected statistical units are predicted for testing (inductive setting). In this setting, persons are rarely connected by the *knows* relations. The *knows* relation in the training and test set is very sparse (0.18%).

Setting 2 also shows the situation where statistical units in the sample are randomly selected, but this time the truth values of statements concerning the statistical units in the training sample are predicted (transductive setting). Some instances of the *knows* relation of the selected statistical units are withheld from training and used for prediction. Prediction should be easier here since the statistics for training and prediction match perfectly.

Setting 3 assumes that the Web address of one user (i.e., one statistical unit) is known. Starting from this random user, users connected by the *knows* relation are gathered by breadth-first crawling and are then added as rows in the training set. The test set is gathered by continued crawling (inductive setting). In this way all profiles are (not necessarily directly) connected and training profiles show a higher connectivity (1.02%) compared to test profiles (0.44%). In this situation generalization can be expected to be easier than in setting 1 and 2 since local properties are more consistent than global ones.

Setting 4 is the combination of settings 2 and 3. The truth values of statements concerning the statistical units in the training sample are predicted (transductive setting). Instances of the *knows* relation are withheld from training and used for prediction.

Settings 5-8 use the same set of statistical units as settings 1-4 respectively. The difference is that in settings 1-4 the data matrix only contains friendship relations to persons in the sample whereas in settings 5-8, the data matrix contains friendship relations any persons in the population. In settings 5-8 we remove those users (friendship attributes) who are known by less than ten users (statistical units), i.e., $\epsilon = 10$. We ended up with a large number of ones in the data matrix when compared to settings 1-4. The concrete numbers of the statistical units and the friendship attributes are shown in *Person* (row) and *Person* (col) respectively in Table 1.

Evaluation Procedure and Evaluation Measure. The task is to predict potential friends of a person, i.e., *knows* statements. For each person in the data set, we randomly selected one *knows* friendship statement and set the corresponding matrix entry to *zero*, to be treated as unknown (test statement). In the test phase we then predicted all unknown friendship entries, including the entry for the test statement. The test statement should obtain a high likelihood value, if compared to the other unknown friendship entries. Here we use the normalized discounted cumulative gain (NDCG) [16] (described in the Appendix) to evaluate a predicted ranking.

Benchmark methods. *Baseline:* Here, we create a random ranking for all unknown triples, i.e., every unknown triple gets a random probability assigned. *Friends of friends in second depth (FOF, d=2):* We assume that friends of friends of a particular person might be friends of that person too. From the RDF graph point of view the *knows* relation propagates one step further alongside the existing *knows* linkages.

4.2 Results

In settings 1 and 2 we randomly sampled 2,000 persons for the training set. In addition, in setting 1 we further randomly sampled 2,000 persons for the test set. In setting 3, 4,000 persons were sampled, where the first half was used for training and the second half for testing. Setting 4 only required the 2,000 persons in the training set. In settings 5-8 we followed the same sampling strategies as in settings 1-4 respectively and extracted all users known by the sampled users to form the friendship attributes. In each case, sampling was repeated 5 times such that error bars could be derived. Table 1 reports details of the samples (training set and, if applicable, test set). The two benchmark methods and the four multivariate prediction approaches proposed in Section 3.4 were then applied to the training set. For each sample we repeated the evaluation procedure as described above 10 times. Since NNMF is only applicable in a transductive setting, it was only applied in setting 1, 3, 5 and 7. Moreover, the *FOF, d=2* is not applicable in settings 5-8, since it is impossible for many statistical units to access the friends of their friends.

Figures 4 and 5 show the experimental results for our FOAF data set. The error bars show the 95% confidence intervals based on the standard error of the mean over the samples. The figures plot the *NDCG all* score of the algorithms against the number of latent variables in settings 1, 2, 5, 6 in Figure 4 and in settings 3, 4, 7, 8 in Figure 5. The best *NDCG all* scores of all algorithms in different settings are shown in Table 2, where r indicates the number of latent variables achieving the best scores.

First, we observe that the experimental results in settings 5-8 are much better than those in settings 1-4. This can be attributed to the fact that in settings 5-8 columns were pruned more drastically and a more dense friendship pattern was achieved.

Another observation is that all four multivariate prediction approaches clearly outperform the benchmark algorithms in all settings, although in settings 1 and 2 NNMF and SVD are only slightly better than FOF, d=2.

Table 1. Number of individuals and number of instantiated relations in the full triple set, in the pruned triple set (see text) and statistics for the different experimental settings

		full	pruned	setting 1		setting 2	setting 3		setting 4	setting 5		setting 6	setting 7		setting 8
				training	test		training	test		training	test		training	test	
Concept	$Person$ (row)	32,062	14,425	2,000	2,000	2,000	2,000	2,000	2,000	2,000	2,000	2,000	2,000	2,000	2,000
#Indivi.	$Person$ (col)	-	-	2,000	2,000	2,000	2,000	2,000	2,000	1,122	1,122	1,122	1,297	1,297	1,297
	$Location$	5,673	320	320	320	320	320	320	320	320	320	320	320	320	320
	$School$	15,744	329	329	329	329	329	329	329	329	329	329	329	329	329
	$Interest$	4,695	118	118	118	118	118	118	118	118	118	118	118	118	118
	$On.ChatAcc.$	5	5	5	5	5	5	5	5	5	5	5	5	5	5
	$Date$	4	4	4	4	4	4	4	4	4	4	4	4	4	4
	$\#BlogPosts$	5	5	5	5	5	5	5	5	5	5	5	5	5	5
Role	*knows*	530,831	386,327	7,311	7,339	7,339	40,786	17,613	40,786	14,909	16,869	16,869	41,705	18,595	41,705
#Inst.	($sparsity$)	0.05%	0.19%	0.18%	0.18%	0.18%	1.02%	0.44%	1.02%	0.66%	0.75%	0.75%	1.61%	0.72%	1.61%
	$residence$	24,368	7,964	1,122	1,106	1,106	1,172	1,217	1,172	1,122	1,106	1,106	1,172	1,217	1,172
	$attends$	31,507	5,088	676	747	747	718	749	718	676	747	747	718	749	718
	has	9,616	1,659	206	246	246	216	208	216	206	246	246	216	208	216
	$holds$	19,021	8,319	1,134	1,087	1,087	1,168	1,075	1,168	1,134	1,087	1,087	1,168	1,075	1,168
	$dateOfBirth$	10,040	5,287	777	715	715	779	784	779	777	715	715	779	784	779
	$posted$	31,959	14,369	1,993	1,992	1,992	1,994	1,991	1,994	1,993	1,992	1,992	1,994	1,991	1,994

Table 2. Best *NDCG all* averaged over samples with 95% confidence interval where r stands for the number of latent variables

Method	setting 1	setting 2	setting 3	setting 4	setting 5	setting 6	setting 7	setting 8
$Baseline$	0.1092 $\pm$ 0.0003	0.1092 $\pm$ 0.0003	0.1094 $\pm$ 0.0001	0.1094 $\pm$ 0.0001	0.1213 $\pm$ 0.0005	0.1213 $\pm$ 0.0005	0.1216 $\pm$ 0.0042	0.1216 $\pm$ 0.0042
$FOF, d = 2$	0.2146 $\pm$ 0.0095	0.2146 $\pm$ 0.0095	0.1495 $\pm$ 0.0077	0.1495 $\pm$ 0.0077	NaN	NaN	NaN	NaN
$NNMF$	NaN	0.2021 $\pm$ 0.0058 r=100	NaN	0.2983 $\pm$ 0.0197 r=150	NaN	0.2864 $\pm$ 0.0067 r=150	NaN	0.3217 $\pm$ 0.0403 r=100
SVD	0.2174 $\pm$ 0.0061 r=150	0.2325 $\pm$ 0.0074 r=100	0.2085 $\pm$ 0.0147 r=200	0.3027 $\pm$ 0.0179 r=100	0.2688 $\pm$ 0.0044 r=150	0.3176 $\pm$ 0.0092 r=150	0.2407 $\pm$ 0.0413 r=100	0.3411 $\pm$ 0.0179 r=50
LDA	0.2514 $\pm$ 0.0049 r=200	0.2988 $\pm$ 0.0057 r=200	0.2288 $\pm$ 0.0123 r=200	0.3374 $\pm$ 0.0117 r=200	0.2640 $\pm$ 0.0022 r=150	0.3359 $\pm$ 0.0079 r=200	0.2331 $\pm$ 0.0143 r=150	0.3470 $\pm$ 0.0372 r=200
$RRPP$	0.2483 $\pm$ 0.0018 r=400	0.2749 $\pm$ 0.0037 r=400	0.2252 $\pm$ 0.0049 r=400	0.3315 $\pm$ 0.0109 r=400	0.2956 $\pm$ 0.0019 r=400	0.3582 $\pm$ 0.0049 r=400	0.2607 $\pm$ 0.0088 r=400	0.3591 $\pm$ 0.0237 r=400

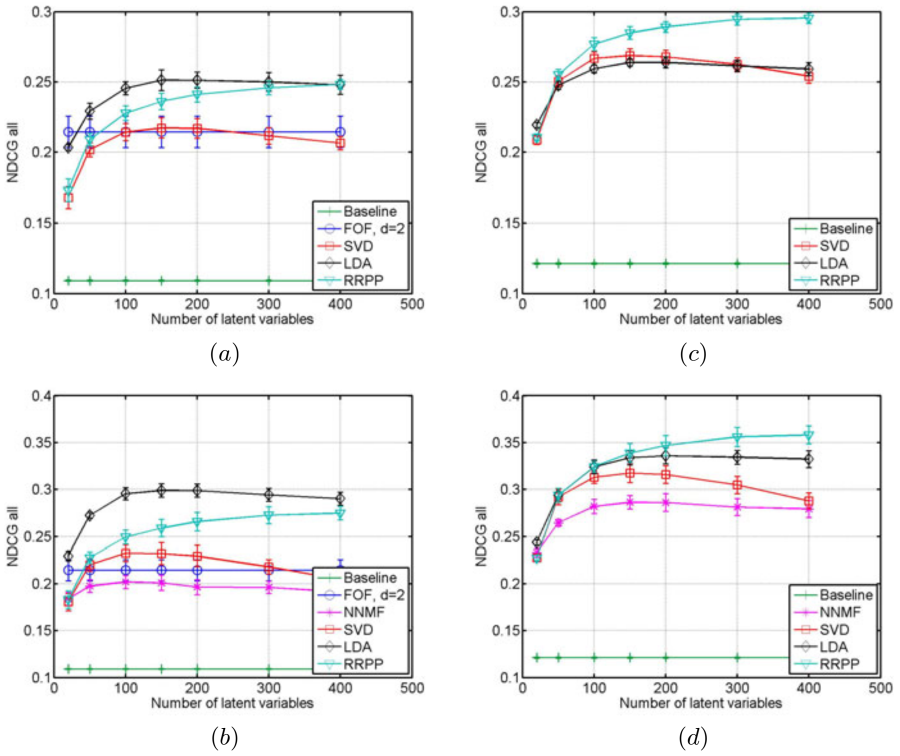


Fig. 4. Comparison between different algorithms. *NDCG all* is plotted against the number of latent variables: (a)-(d) for settings 1, 2, 5, 6 respectively.

Furthermore, we observe that LDA and RRPP outperform NNMF and SVD in each setting, and that LDA and RRPP are not sensitive to the number of latent variables as long as the chosen number is reasonably high. LDA reaches its maximum NDCG score, for instance, with $r = 150$ latent variables in setting 4 and the performance does not deteriorate when the number of latent factors is increased. The score of RRPP keeps increasing and does not drop down in the observed range of the number of latent variables. In contrast, NNMF and SVD are sensitive with respect to the predefined number of latent variables.

Comparing the results over different settings we can observe that for the multivariate prediction approaches one obtains best performance in setting 4, next best performance in setting 2, then follows setting 1 and 3 is the most difficult setting. The corresponding order can be seen in settings 5-8. The baseline method, random guess, is independent to the settings and achieves almost the same score in all settings. The fact that the scores in settings 4 and 8 are the best indicates that a link-following sampling strategy in general gives better performance than random sampling. Similar results in statistical comparisons between random and link-following sampling have been obtained in other works, e.g., [17].

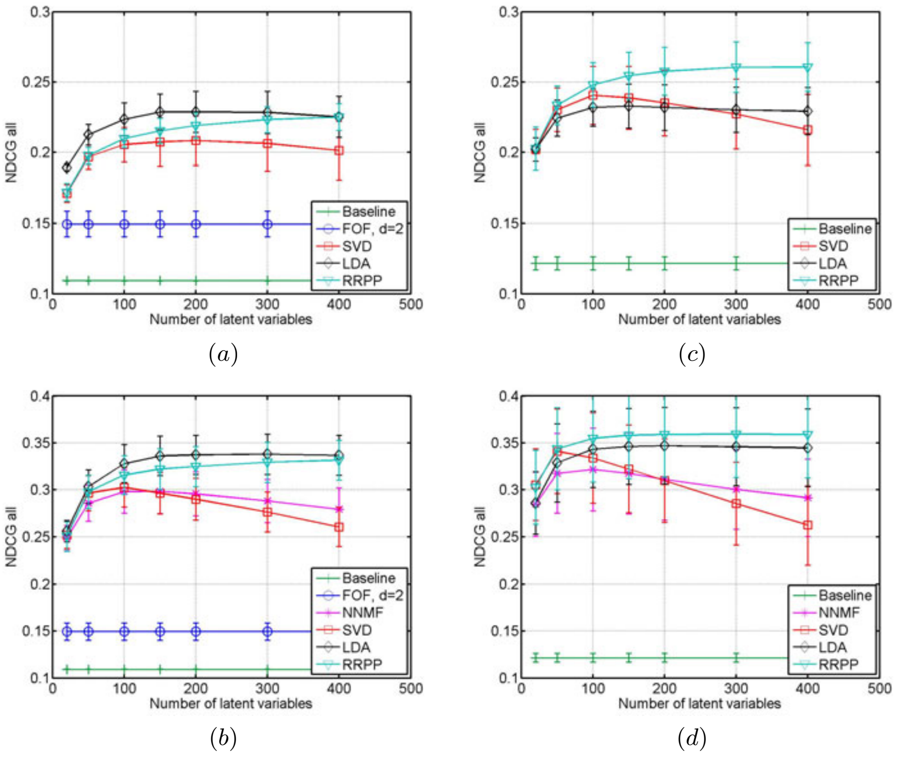


Fig. 5. Continue Figure 4: (a)-(d) for settings 3, 4, 7, 8 respectively

Finally, we observe that the prediction performance in setting 1 is only slightly worse than the prediction performance in setting 2, while the prediction performance in setting 4 is much better than in setting 3. This phenomenon occurs in settings 5-8 too. We attribute this to the general statistics in the training and the test set which are very different both in setting 3 and setting 7. In Table 1 it is apparent that for instance, in setting 3 the *knows* relation in the training data set (1.02%) is significantly more dense than in the test data set (0.44%). Intuitively speaking, the people in the training know each other quite well, but the people in the test do not know the people in the training as much.

5 Conclusions and Outlook

The paper describes extensions to the SUNS approach introduced in [12]. The SUNS approach is based on multivariate prediction which is quite suitable for the typical SW data situation. In our experiments based on the FOAF data set, LDA and RRPP showed best performance, and the performance is insensitive to the rank of the approximation, resp. to the number of latent variables. This can be explained by the fact that LDA, in contrast to NNMF, is a Bayesian approach and by the fact that the RRPP, in contrast to SVD, is regularized. Thus LDA or RRPP can be default methods being insensitive to exact parameter tuning. All four approaches exploited the benefits of multivariate prediction since approaches based on single predictions (not reported here) did not even reach the performance of the benchmark approaches.

The proposed approach can be extended in many ways. One might want to allow the user to specify additional parameters in the learning process, if desired, along the line of the extensions described in [2]. Another extension concerns ontological background knowledge. So far, ontological background knowledge was considered by including logically inferred statements into learning. Ongoing work explores additional ways of exploiting ontological background information, e.g., for structuring the learning matrix.

Finally we want to demonstrate how learned probabilistic statements can be queried. The following SPARQL query illustrates a query for LiveJournal users who live in Munich and might want to be Trelena's friend:

```
PREFIX ya:   http://blogs.yandex.ru/schema/foaf/
PREFIX foaf: http://xmlns.com/foaf/0.1/
PREFIX dc:   http://purl.org/dc/elements/1.1/
SELECT DISTINCT ?person
WHERE
 {?person ya:located ?city .
  ?person foaf:knows <http://trelana.livejournal.com/trelana>
          WITH PROB ?prob .
   FILTER REGEX(?city, "Munich") .
 }
ORDER BY DESC(?prob)
```

The query includes the predicted *knows* triples for Trelena and rates them by predicted probability.

Acknowledgements. We acknowledge funding by the German Federal Ministry of Economy and Technology (BMWi) under the THESEUS project and by the EU FP 7 Large-Scale Integrating Project LarKC.

References

1. Raedt, L.D., Jaeger, M., Lee, S.D., Mannila, H.: A theory of inductive query answering. In: ICDM (2002)
2. Kiefer, C., Bernstein, A., Locher, A.: Adding data mining support to SPARQL via statistical relational learning methods. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 478–492. Springer, Heidelberg (2008)
3. Getoor, L., Friedman, N., Koller, D., Pferrer, A., Taskar, B.: Probabilistic relational models. In: Getoor, L., Taskar, B. (eds.) Introduction to Statistical Relational Learning. MIT Press, Cambridge (2007)
4. Domingos, P., Richardson, M.: Markov logic: A unifying framework for statistical relational learning. In: Getoor, L., Taskar, B. (eds.) Introduction to Statistical Relational Learning. MIT Press, Cambridge (2007)
5. Xu, Z., Tresp, V., Yu, K., Kriegel, H.P.: Infinite hidden relational models. In: Uncertainty in Artificial Intelligence (UAI) (2006)

6. Kemp, C., Tenenbaum, J.B., Griffiths, T.L., Yamada, T., Ueda, N.: Learning systems of concepts with an infinite relational model. In: Poceedings of the National Conference on Artificial Intelligence, AAAI (2006)
7. Quinlan, J.R.: Learning logical definitions from relations. Machine Learning 5(3) (1990)
8. Muggleton, S., Feng, C.: Efficient induction of logic programs. In: Proceedings of the 1st Conference on Algorithmic Learning Theory, Ohmsma, Tokyo (1990)
9. De Raedt, L.: Attribute-value learning versus inductive logic programming: The missing links (extended abstract). In: Page, D.L. (ed.) ILP 1998. LNCS, vol. 1446, Springer, Heidelberg (1998)
10. Lavrač, N., Džeroski, S., Grobelnik, M.: Learning nonrecursive definitions of relations with LINUS. In: EWSL 1991: Proceedings of the European Working Session on Learning on Machine Learning (1991)
11. Tresp, V., Yu, K.: Learning with dependencies between several response variables. In: Tutorial at ICML (2009)
12. Tresp, V., Huang, Y., Bundschus, M., Rettinger, A.: Materializing and querying learned knowledge. In: Proceedings of the First ESWC Workshop on Inductive Reasoning and Machine Learning on the Semantic Web (2009)
13. Lee, D.D., Seung, H.S.: Learning the parts of objects by non-negative matrix factorization. Nature (1999)
14. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. J. Mach. Learn. Res. 3 (2003)
15. Brickley, D., Miller, L.: The Friend of a Friend (FOAF) project, http://www.foaf-project.org/
16. Jarvelin, K., Kekalainen, J.: IR evaluation methods for retrieving highly relevant documents. In: SIGIR 2000 (2000)
17. Neville, J., Gallagher, B., Eliassi-Rad, T.: Evaluating statistical tests for within-network classifiers of relational data. In: ICDM 2009 (2009)

Appendix

Details on the NDCG Score

We use the normalized discounted cumulative gain (NDCG) to evaluate a predicted ranking. NDCG is calculated by summing over all the gains in the rank list R with a log discount factor as

$$\mathrm{NDCG}(R) = \frac{1}{Z}\sum_k \frac{2^{r(k)} - 1}{\log(1+k)},$$

where $r(k)$ denotes the target label for the k-th ranked item in R, and r is chosen such that a perfect ranking obtains value 1. To focus more on the top-ranked items, we also consider the *NDCG@n* which only counts the top n items in the rank list. These scores are averaged over all ranking lists for comparison.

Learning Action Descriptions of Opponent Behaviour in the Robocup 2D Simulation Environment*

A. Illobre, J. Gonzalez, R. Otero, and J. Santos

Computer Science Department, University of Corunna, Spain
{infjdi00,jgonzalezi,otero,santos}@udc.es

Abstract. The Robocup 2D simulation competition [13] proposes a dynamic environment where two opponent teams are confronted in a simplified soccer game. All major teams use a fixed algorithm to control its players. An unexpected opponent strategy, not previously considered by the developers, might result in winning all matches. To improve this we use ILP to learn action descriptions of opponent players; for learning on dynamic domains, we have to deal with the frame problem. The induced descriptions can be used to plan for desired field states. To show this we start with a simplified scenario where we learn the behaviour of a goalkeeper based on the actions of a shooter player. This description is used to plan for states where a goal can be scored. This result can directly be extended to a multiplayer environment.

Keywords: ILP, Action Descriptions, Answer Sets, Nonmonotonic Reasoning, Robocup Simulation Environment.

1 Introduction

RoboCup [13] is an international joint project created to promote AI and robotics research, by providing a standard problem (a simplified soccer game) where different technologies can be integrated and examined. The RoboCup competition includes real robot competitions and a software agent competition. In this last case, the soccer players must interact with a simulated environment based on incomplete and partially incorrect data[1].

The RoboCup simulated league is based on *rcssserver*, a soccer simulation environment. In *rcssserver*, a match is carried out in a client/server style: the server provides a virtual field and simulates all movements of a ball and players, while each client controls the movements of one player from a partial perception of its environment (referee messages, player body status, etc.). Time is discretized in *simulation cycles*. The server introduces noise in the visual sensor data as well as in the movement of objects and parameters of commands.

* This paper has been funded by the Ministry of Science and Education of Spain through project TIN2007-64330.

[1] http://sserver.sourceforge.net/wiki/index.php/Main_Page

Currently teams use fixed strategies to control its players. If an opponent uses an unexpected strategy, then the controlled team might show a low performance. To overcome this limitation, we propose a technique to learn action descriptions for opponent players, based on the actions performed by the team. This allows for planning for desired field states (for instance, a state where a goal can be scored). This means that an optimal, specific strategy will be available for each opponent team.

To test the proposed method, we solve the problem of placing a shooter player in a field position where a goal can be scored, in an environment with an opponent goalkeeper. This is achieved in two steps. First, a model to decide if a goal can be scored from a given field state is proposed, based on the position of the shooter player and the opponent goalkeeper. Then, an action description for the movement of the opponent goalkeeper is learned. The resulting model can be used to plan for optimal shooting situations.

This requires to learn in a dynamic domain, so it is suitable to solve the frame problem. We do so by applying the method proposed on [11], using the system IAction [12].

The paper is organized as follows. Section 2 introduces the concept of *action description*, and briefly describes the method [11] for learning on dynamic domains. Section 3 proposes a solution to automatically model the behaviour of opponent players, exemplifying it on the award-winning team CMUnited99 [14] goalkeeper. Finally, section 4 briefly reviews previous work and presents the conclusions.

2 Action Descriptions for Dynamic Domains

We follow the method on [11] that represents dynamic domains using action descriptions. An *action description* represents the properties of a dynamic domain as *fluents*. These properties will change depending on performed *actions*. For example, shooting the ball in the correct direction will increase the score count. Both, actions and fluents, will be represented as predicates.

Each step on the evolution of the dynamic domain is represented by a *situation*. A *narrative* represents a particular example of the evolution of the domain from an initial situation s_0 to a final situation s_n. Different narratives represent different examples of the evolution of the environment, depending on the initial state of the environment and the actions performed.

An action description can be represented using several kinds of rules. *Action laws* take the following form:

```
e(S,N) :- a(S,N), prev(S,PS), PRECOND(PS,N).
```

where $e(S, N)$ is a fluent and $a(S, N)$ is an action. S and PS are situation variables, N represents a narrative. $prev(S, PS)$ defines situation S as the successor of the situation PS. $PRECOND(PS, N)$ represents a conjunction of fluents $f_1(PS, N), ..., f_n(PS, N)$ and might be missing.

Intuitively, these rules state that when action a is performed on situation S and fluents $f_1, ..., f_n$ are true on the previous situation PS, then the effect fluent e will be true on the situation S.

Frame axioms take the following form:

```
e(S,N) :- a(S,N), prev(S,PS), e(PS,N), PRECOND(PS,N).
```

Frame axioms are needed to explain the target fluent e when it persists from the previous situations. Both type of rules are needed to cover the examples. We need a frame axiom for each combination of fluent and action. When a domain is described with frame axioms it is said that we have the frame problem. To solve the frame problem we need a compact description of the persistence of the fluents. This is solved on Stable Models [6] using inertia axioms instead of frame axioms. Two single inertia axioms for a fluent replace all frame axioms with such fluent in the head.

Inertia axioms take the following form:

```
e(S,N) :- prev(S,PS), e(PS,N), not ne(S,N).
ne(S,N) :- prev(S,PS), e(PS,N), not e(S,N).
```

Where $ne(S, N)$ is the literal complementary to e and *not* represents negation as failure (NAF). The first rule should be read as: the fluent e will be true on S if e is true on the previous situation PS and we cannot prove that e is false on S.

2.1 Efficient Induction of Action Laws with IAction

We use the method [11] to efficiency learn on dynamic domains. This method uses causality to get solutions to the frame and ramification problems in induction. Causality allows to translate the nonmonotonic induction problem to a form where monotonic methods of ILP can be used to provide a complete method for induction of action descriptions without the frame problem. Without this the methods defined on monotonic formalisms will have the frame problem. A possible alternative solution would be to apply some existing restricted nonmonotonic induction methods, but it is not known how to use these methods for learning without the frame problem.

A nonmonotonic induction method would include the inertia axioms in the background to provide solutions free from the frame problem. On monotonic ILP methods the frame axioms are induced to cover the fluents on the examples that persist. Nevertheless we could avoid the induction of the fluents that persist by not providing those persistent examples and selecting, as a target, the caused fluents only.

Consider a simplified Robocup domain with one player. There are two fluents in this domain: *goal* (resp. *ngoal*) represents that a goal has been scored and *hasball* (resp. *nhasball*) represents that the player has the ball. Three actions are considered: *shoot*, *wait* (do nothing), *getball*. We want to learn how to score a goal. Table 2.1 shows a possible narrative for this domain. Note that the target is missing for situation 3. This is called a *missing segment*. A missing segment

Table 1. Transformation of an input narrative by method [11]

situation	0	1	2	3	4
goal	*ngoal*	*ngoal*	*ngoal*	?	*goal*
hasball	*nhasball*	*nhasball*	*nhasball*	*hasball*	*hasball*
actions		$shoot(1,1)$	$wait(2,2)$	$getball(m34,3)$	$shoot(m34,4)$
$causedE^{+}$				$pgoal(m34)$	
$causedE^{-}$	*npgoal*	*npgoal*	*npgoal*		

represents that there is a positive example on causality on some situation inside it, similar cases have already been considered in machine learning under Multiple Instance (MI) learning.

Missing example instances in the target predicate are common in learning. It is precisely because of these missing instances that alternative solutions to induction provide different generalizations; in this sense, more missing instances allow more generalization in the solution. Induction in action seems to behave the other way around, missing examples instead of facilitating induction turn it more complex. Note also that there is no problem with missing examples when induction is directly applied on a monotonic ILP method. But recall also that all these solutions have the frame problem. This points out the close relationship between dealing with missing examples and solving the frame problem in induction.

To be able to efficiently learn action rules without the frame problem a transformation on the input is defined.

Step 1: For every fluent f on the domain, add the constraint $:-f(S), nf(S)$. This representation avoids the CWA of LP for fluents and allows the reference to the negative fluent without using NAF, thus inside definite LP. The CWA for fluents is not interesting, because when some fluent instance cannot be proved it is better to assume it persisted than to assume it is false.

Step 2: Define an extra argument for every action, e.g. *shoot(ES,S)*, being *ES* the new argument and S the argument defining the situation. Define a new constant (e.g. *m34*) for every action on the missing segment and the immediate situation after the missing segment. For any other action the *ES* argument is that of the situation, e.g. *wait(2,2)*.

Step 3: For every missing segment with complementary target instances at the situations immediately before and after the segment, a caused instance is defined as follows: If the situation inmediately after the segment has a fluent f (resp. *nf*), define *pf(es)* (resp. *pnf(es)*), where *es* is the constant name of the missing segment. A single instance of causality is extracted from a missing segment, and the instance is at the segment *es* as a whole.

Step 4: For the target fluent *f(S)* define the fluent *npf(S)* (also *npnf(S)* for *nf(S)*).

Step 5: Apply a complete monotonic induction method of ILP, e.g. IE, with target fluent *pf(S)* instead of the original target *f(S)*, e.g. *pgoal* instead of *goal.*

The causality of the target fluent will be induced in the form of action laws. In our example this action law will be induced: $pgoal(ES) : -shoot(ES, S), prev(S, PS), getball(PS)$.

A simple transformation provides descriptions of actions in Stable Models [6]. For every induced causal action law put the head directly on the original target *f(S)*, instead of the causal fluent *pf(S)*: $f(S) : -prev(S, PS), f(PS), not\ nf(S)$. Complete the solution adding the inertia axioms for *f(S)* and *nf(S)*.

3 Modeling Opponent Behaviour

To solve the problem of adapting to specific opponent strategies, we propose to learn an action description on opponent behaviour. This action description can then be used as input to a suitable non-monotonic formalism (e.g. *Stable Models* [4]) to solve planning and prediction problems on the domain.

To exemplify this approach, a simplified scenario with an opponent goalkeeper and a shooter player is proposed. The objective is to model the behaviour of the goalkeeper based on the behaviour of the player. This is, we want to learn action rules for modeling how the goalkeeper is going to move, depending on the actions of the player. This action description is then combined with a generic action description describing the effects of the shooter actions on the possibility of scoring a goal. The resulting theory can be used plan for states where a goal can be scored.

3.1 Discretization of the Environment

The values of the properties of the field in *rcssserver* are represented as real numbers. We discretize the positions and angles with a precision of 10^{-1}: the field is divided in approximately 420000 squares.

Because we do not know the internal state of the oponent player it is possible, from the point of view of the shooter agent, that there are two different situations s_i and s_j where the field state S and the action taken a are the same but the effects of the action are different. So we cannot learn an action rule for one of the situations without being inconsistent with the other. For dealing with these non-deterministic effects we remove from the evidence all conflicting situations but that with the highest probability, estimated from the available examples. This preprocessing will remove only a 2% of the evidence that will be used in this work.

3.2 Representation

In the movement model used by *rcssserver*, a player moves by performing a *dash* action that results in an acceleration. The player speed is only modified by a small decay on each simulation cycle or or by another dash action. If the objective of a player is to move to a certain point, then it will apply dash actions periodically to maintain its speed. Thus, we represent the movement of an opponent player as the distance covered after applying a certain action. By default, an opponent player will maintain its speed and direction.

To explain changes on the player speed, a simplified set of possible shooter actions, based on that required by *rcssserver*, will be used. The selected fluents encode the position and direction of the players in the field. Every predicate representing a fluent or an action will have at least a term S (situation) and a term N (narrative). For example, the action $move(s3, n4)$ represents that the shooter will perform the action *move* on the situation 3 of the narrative 4.

The background used for the learning task is defined using the set of actions and fluents described below.

Actions:
- $move(S, N)$: The shooter moves on its current direction at maximum speed.
- $turn(S, N, right/left)$: The shooter rotates clockwise/counterclockwise 45°.
- $wait(S, N)$: The shooter stays still.

Fluents:
- $at(P, S, N, X, Y)$: player P is at position (X, Y). [0,0] represents the center of the field. Positions left or down the center take negative values.
- $angle(P, S, N, A)$: angle of player P is A.
- $speedx(P, S, N, V_x)$: horizontal speed of the player X is V_x. Positive values represent movements to the right, negative values represents movements to the left. Speed range is [-1.1,1.1], currently discretized in 23 values.
- $speedy(P, S, N, V_y)$: vertical speed of the player X is V_y. Positive values represent upward movements, negative values represent downward movements.

Given a simplified environment with one goalkeeper (team 1) and a player (team 2), we want to learn how the goalkeeper will move (direction and speed) depending on the movements of the player on the field. The target predicates will be the speed and direction of the goalkeeper on the x-axis (*speedx*) and on the y-axis (*speedy*). The constant *goalie* represents the goalkeeper of the opposite team and the constant *shooter* represents the shooter player. Negative examples represent that a goalkeeper can only move on the current direction with a speed classified in the same discretization value.

Evidence on goalkeeper behaviour has been extracted by developing a custom shooter player. A test set of 2000 narratives has been generated.

3.3 Learning Results

The following is an example of the learned rules.

```
speedx(goalie,S,N,-0.9) :- turn(S,N,left), prev(S,PS), at(shooter,S,N,X,Y),
   lessequal(X,30.0), lessequal(Y,10.0), greaterequal(Y,-10.0),
   at(goalie,S,N,X2,Y2), lesequal(X2,44.5), angle(shooter,S,N,A),
   greaterequal(A,-45), lessequal(A,45).
```

The intuitive meaning of this rule is that the goalkeeper moves forward if the player turns his back against the goal, on a certain distance.

The learning process also produces the required inertia axioms for the learning target. For example:

```
speedx(P,S,N,V) :- prev(S,PS), speed(P,PS,N,V), not -speed(P,S,N,V).
```

Meaning that the horizontal speed is assumed to persist unless an action causes it to change.

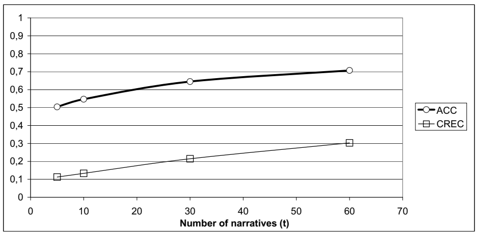


Fig. 1. Performance of goalie speed prediction

Validation. To assess the performance of this proposal, the following metrics have been defined: (1) *Accuracy (ACC):* proportion of cases where, given an state and action, the speed of the opponent player is predicted correctly, (2) *Change recall (CREC):* proportion of cases where, given an state and action for which the speed is expected to change, the new speed value is predicted correctly.

Metric values are calculated as follows. A set of narratives of size t is randomly generated, and an action description learned from it. Then, this description is tested again the test set. Figure 1 shows the results of this validation. Standard error is within ± 0.02 for all cases.

The results show that the method achieves high levels of accuracy with restricted evidence and time. This performance is achieved because of the chosen default assumption on the goalkeeper speed: the speed will be the same unless an action causes it to change. The accuracy on predicting speed change, however, is lower. This indicates the need for more evidence and learning time for the method to make more significant predictions on the changes of the goalkeeper speed. An enhanced representation for the background information (for instance, considering distances between players and key points of the field, such as the goal) could allow for a more compact description and more general rules, thus achieving higher accuracy values under the same restrictions on input evi- dence size and learning time. This is proposed as future work.

Also, the movement of the opponent goalkeeper might depend on the position of other opponent players. The method can be directly extended to this multiplayer environment, by encoding the position of other players using the representation explained above. Preliminary results with the CMUnited99 goalkeeper in presence of two other opponent players, show that rules using the new information can be found.

4 Previous Work and Conclusions

There are several works for inducing the effects of actions on dynamic domains. Moyle and Muggleton [9] study the induction of Event Calculus programs, a particular action formalism. The methods proposed there are different to [11] and rely on working with negation as failure in the background (for inertia) during induction. Benson [1] describes the system TRAIL. Given an action and a possible effect on the domain (called Teleo-Operator), TRAIL tries to find a set of preconditions under which that actions causes that effect. Later, it uses this information to build a Teleo-Reactive tree to decide which actions should be applied to achieve a certain goal.

There are also several proposals for learning action strategies in dynamic domains, some of which have been applied to the Robocup Simulation Environment. Driessens and De Raedt [3] learn clauses with actions in the head to describe the behaviour of a preexisting team in order to validate its implementation. Torrey and Maclin [15] use ILP to learn clauses with actions in the head as a step for transferring knowledge between reinforcement learning tasks. Khardon [5] proposes to learn production rules representing action strategies.

In other line of work, Matsui et al. [7] use ILP to learn rules to decide if a certain action will be successful for a certain field state.

In this paper we have used the method on [11] to learn action descriptions on the behaviour of opponent players in the Robocup Simulation Environment, studying the case of predicting the movement of a goalkeeper based on the actions of a shooter player. With these descriptions, a suitable non-monotonic formalism, like Stable Models [4], can be used to plan for desired states. Current ILP methods are not suitable for learning in dynamic domains: most of the solutions have the frame problem. Results show that the proposed method has the potential to reach high accuracy values.

References

1. Benson, S.: Learning action models for reactive autonomous agents. Tech. rep. CS-TR-97-1589, Ph. The., Department of Computer Science, Stanford University (1997)
2. Chen, M., et al.: RoboCup Soccer Server, Users Manual (2002)
3. Driessens, K., Jacobs, N., Cossement, N., Monsieurs, P., De Raedt, L.: Inductive verification and validation of the kULRoT roboCup team. In: Asada, M., Kitano, H. (eds.) RoboCup 1998. LNCS (LNAI), vol. 1604, pp. 193–206. Springer, Heidelberg (1999)
4. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: Proceedings of the Fifth International Conference on Logic Programming, ICLP (1988)
5. Khardon, R.: Learning to take Actions. Machine Learning 35(1) (1999)
6. Lifschitz, V.: Action languages, answer sets and planning. In: The Logic Programming Paradigm: a 25-Year Perspective. Springer, Heidelberg (1999)
7. Matsui, T., Inuzuka, N., Seki, H.: Adapting behaviour by inductive prediction in soccer agents. In: Mizoguchi, R., Slaney, J.K. (eds.) PRICAI 2000. LNCS, vol. 1886, p. 807. Springer, Heidelberg (2000)
8. McCarthy, J., Hayes, P.J.: Some philosophical problems from the standpoint of artificial intelligence. Machine Intelligence 4, 463–502 (1969)
9. Moyle, S., Muggleton, S.H.: Learning programs in the event calculus. In: Džeroski, S., Lavrač, N. (eds.) ILP 1997. LNCS, vol. 1297. Springer, Heidelberg (1997)
10. Muggleton, S.H.: Inverse entailment and Progol. New Generation Computing 13, 245–286 (1995)

11. Otero, R.: Embracing Causality in Inducing the Effects of Actions. In: Conejo, R., Urretavizcaya, M., Pérez-de-la-Cruz, J.-L. (eds.) CAEPIA/TTIA 2003. LNCS (LNAI), vol. 3040, pp. 291–301. Springer, Heidelberg (2004)
12. Otero, R., Varela, M.: IAction, a System for Learning Action Descriptions for Planning. In: ILP (2006)
13. RoboCup Official Site, http://www.robocup.org
14. Stone, P., Riley, P., Veloso, M.: The CMUnited-99 champion simulator team. AI Magazine 21(3), 33–40 (2000)
15. Torrey, L., Shavlik, J., Walker, T., Maclin, R.: Skill acquisition via transfer learning and advice taking. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 425–436. Springer, Heidelberg (2006)

Hypothesizing about Causal Networks with Positive and Negative Effects by Meta-level Abduction*

Katsumi Inoue[1], Andrei Doncescu[2], and Hidetomo Nabeshima[3]

[1] National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
[2] LAAS-CNRS UPR 8001
Avenue du Colonel Roche, 31007 Toulouse, France
[3] Division of Medicine and Engineering Science, University of Yamanashi
4-3-11 Takeda, Kofu, Yamanashi 400-8511, Japan

Abstract. Meta-level abduction discovers missing links and unknown nodes from incomplete networks to complete paths for observations. In this work, we extend applicability of meta-level abduction to deal with networks containing both positive and negative causal effects. Such networks appear in many domains including biology, in which inhibitory effects are important in signaling and metabolic pathways. Reasoning in networks with inhibition is inevitably nonmonotonic, and involves default assumptions in abduction. We show that meta-level abduction can consistently produce both positive and negative causal relations as well as invented nodes. Case studies of meta-level abduction are presented in p53 signaling networks, in which causal rules are abduced to suppress a tumor with a new protein and to stop DNA synthesis when damage is occurred.

1 Introduction

Abduction and induction are both ampliative reasoning, and play essential roles in knowledge discovery and development in science and technology. The use of prior or *background knowledge* in scientific applications has directed our attention to *theory completion* [12] rather than classical learning tasks such as concept learning and classification. There, abduction is used to complete proofs of observations from *incomplete* background knowledge. In theory completion, the larger the knowledge base becomes, the more inference steps are required.

In scientific domains, background knowledge is often structured in a *network* form. In biology, a sequence of signalings or biochemical reactions constitutes a network called a *pathway*, which specifies a mechanism to explain how genes or cells carry out their functions. However, information of biological networks in public-domain databases is generally incomplete in that some details of reactions, intermediary genes/proteins or kinetic information are either omitted or undiscovered. To deal with incompleteness of pathways, we need to predict the status of relations which is consistent with the status of nodes [27,29,8,21], or insert missing arcs between nodes to explain observations

* This research is supported in part by the 2008-2011 JSPS Grant-in-Aid for Scientific Research (A) No. 20240016.

[30,10,28,1]. These goals are characterized by abduction as theory completion, in which status of nodes or missing arcs are added to account for observations.

A method to discover unknown relations from incomplete networks has been introduced in [6] based on *meta-level abduction*. Given a network representing causal relations, called a *causal network*, missing links and nodes are abduced in the network to account for observations. The method can be implemented in SOLAR [14,15], an automated deduction system for consequence finding, using a first-order representation for algebraic properties of causality and the full-clausal form of network information and constraints. Meta-level abduction by SOLAR is powerful enough to infer missing rules, missing facts, and unknown causes that involve *predicate invention* [13] in the form of existentially quantified hypotheses. In [6], meta-level abduction has been applied to discover physical skills in cello playing examples. and a thorough experimental analysis with a variety of problem instances has been presented in [15, Table 3]. However, all those examples of meta-level abduction in [6,15] contain only one kind of causal effects, which are positive, and it was left open how to deal with *both positive and negative effects*. Then, in this work, we extend applicability of meta-level abduction to deal with networks expressing both positive and negative causal effects. Such networks are often used in biological domains, where *inhibitory effects* are essential in gene regulatory, signaling and metabolic networks.

We will present axioms for meta-level abduction to produce both positive and negative causal relations as well as newly invented nodes. Reasoning in networks with inhibition is inevitably *nonmonotonic*, and involves default assumptions in abduction. Then, applications to p53 signal networks [19,28] are presented as case studies of our framework, in which meta-level abduction discovers theories explaining how tumor suppressors work and how DNA synthesis stops. Such abstract *signaling networks*, although simple, provide one of the most fundamental inference problems in Systems Biology: Given an incomplete causal network, infer possible connections and functions of the target gene/protein. Meta-level abduction in this paper is crucial for this task: First, suggestion of possible additions to prior networks enables scientists to conduct hypothesis-driven experiments with those focused cases. Second, it is quite hard to observe activity levels or quantities of proteins in living organisms [1].

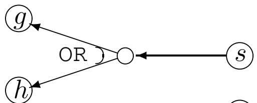
The rest of this paper is organized as follows. Section 2 offers the essential and new perspectives of meta-level abduction and its use for rule abduction. Section 3 then extends meta-level abduction to allow for two types of causal effects, in which positive and negative rules are called *triggers* and *inhibitors*, respectively. Section 4 presents case studies of meta-level abduction applied to completion of sub-networks in p53 signal networks. Section 5 discusses related work, and Section 6 gives a summary and future work.

2 Meta-level Abduction

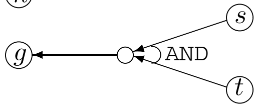
This section revisits the framework for *meta-level abduction* [6].

We suppose a background theory represented in a network structure called a *causal graph* or a *causal network*. A causal graph is a directed graph representing causal relations, which consists of a set of *nodes* and *(directed) arcs* (or *links*).[1] Each node in a causal graph represents some event, fact or proposition. A *direct causal relation* corresponds to a directed arc, and a *causal chain* is represented by the reachability between two nodes. The interpretation of a "cause" here is kept rather informal, and just represents the connectivity, which may refer to a mathematical, physical, chemical, conceptual, epidemiological, structural, or statistical dependency [17]. Similarly, a "direct cause" here simply represents the adjacent connectivity, while its effect is direct only relative to a certain level of abstraction.

We then consider a first-order language to express causal networks. Each node is represented as a *proposition* or a (ground) atom in the language. When there is a direct causal relation from a node s to a node g, we define that $connected(g, s)$ is true. Note that $connected(g, s)$ only shows that s is one of possible causes of g, and thus the existence of $connected(g, t)$ $(s \neq t)$ means that s and t are alternative causes for g. The fact that a direct causal link cannot exist from s to g is represented in an *(integrity) constraint* of the form $\leftarrow connected(g, s)$, which is equivalent to the formula $\neg connected(g, s)$. If a direct causal relation from s has *nondeterministic effects* g and h, it is represented in a disjunction of the form (1). On the other hand, the relation that "g is *jointly* caused by s and t", written intuitively as $(g \leftarrow s \wedge t)$ in the object level, is expressed in a disjunction of the form (2) at the meta level, cf., $(g \leftarrow s \wedge t) \equiv (g \leftarrow s) \vee (g \leftarrow t)$.



$$connected(g, s) \vee connected(h, s) \tag{1}$$



$$connected(g, s) \vee connected(g, t) \tag{2}$$

A complex relation of the form $(g \vee h \leftarrow s \wedge t)$ can be decomposed into two relations, $(s\text{-}t \leftarrow s \wedge t)$ and $(g \vee h \leftarrow s\text{-}t)$, where $s\text{-}t$ represents the intermediate complex. Any other direct causal relation in a causal network can be represented in this way using intermediate complexes and combinations of positive and negative $connected$ literals and disjunctions of the form (1) and (2).

Expression of causal networks is thus done at the *meta level* using the *meta-predicate* $connected$. In this way, (i) each literal in the object level is represented as a term in the meta level, and (ii) each rule in the object level is represented as a (disjunctive) fact in the meta level. The point (ii) can not only hold for rules given in the axioms, but can also be applied to express *inferred rules* at the meta level. Now, to express inferred rules, we introduce another meta-predicate $caused$. For object-level propositions g and s, we define that $caused(g, s)$ is true if there is a *causal chain* from s to g. Then, the causal chains are defined transitively in terms of $connected$ as:

$$caused(X, Y) \leftarrow connected(X, Y). \tag{3}$$

$$caused(X, Y) \leftarrow connected(X, Z) \wedge caused(Z, Y). \tag{4}$$

[1] Precisely speaking, our causal networks bring us more information than directed graphs since negation, disjunctive effects and joint causes are all represented in a network.

Other algebraic properties as well as some particular constraints (e.g., $\neg caused(a, b)$) can also be defined if necessary. Variables in object-level expressions like $g(T)$ and $s(T)$ can be allowed in the meta-level expression like $connected(g(T), s(T))$.

Reasoning about causal networks is realized by deduction and abduction from the meta-level expression of causal networks together with the axioms for causal relations including (3) and (4). In deduction, if a meta-level expression of the form $caused(g, s)$ for some facts g and s can be derived, it means that the rule $(g \leftarrow s)$ can be derived at the object level [6, Section 3.3]. In abduction, if an *observation* O is given as a causal chain $caused(g, s)$, which corresponds to the object-level rule $(g \leftarrow s)$, we want to explain why or how it is caused. Here, g and s are called the *goal fact* and the *source fact*, respectively. O can be given as either a real observation (called an *empirical rule*) or a virtual goal to be achieved. An abductive task is then to discover *hidden rules* that establish a connection from s to g by filling the gaps in causal networks.

Logically speaking, a *background theory* B consists of the meta-level expression of a causal network and the axioms for causal relations in the meta-level containing (3) and (4). When B is *incomplete*, there may be no path between g and s in B, that is, $caused(g, s)$ cannot be derived from B. Then, abduction infers an *explanation* (or *hypothesis*) H consisting of missing relations (links) and missing facts (nodes). This is realized by setting the *abducibles* Γ, the set of candidate literals to be assumed, as the atoms with the predicate *connected*: $\Gamma = \{connected(\_, \_)\}$. A set H of instances of elements of Γ is an *explanation* of O if $B \cup H \models O$ and $B \cup H$ is consistent. An explanation H of O is *minimal* if it does not imply any explanation of O that is not logically equivalent to H. Minimality of explanations in meta-level abduction corresponds to minimal additions in causal graphs, and are reasonable according to the principle of Occam's razor. For example, suppose the observation $O = caused(g, s) \wedge caused(h, s)$, that is, the multiple causal chains between two goal facts g, h and the source fact s. Examples of minimal explanations of O containing one intermediate node are as follows.

$$H_1 : \exists X(connected(g, X) \wedge connected(h, X) \wedge connected(X, s)),$$
$$H_2 : \exists X(connected(g, X) \wedge connected(X, h) \wedge connected(h, s)).$$

Here, H_1 corresponds to the three rules $\{(g \leftarrow \chi), (h \leftarrow \chi), (\chi \leftarrow s)\}$, hence rule abduction is realized here. Moreover, these hypotheses contain existentially quantified variables, where χ can be regarded as either some existing node or a new unknown node. In this way, *predicate invention* [13] is partially realized in meta-level abduction. As H_1 and H_2 represent different connectivities, we can enumerate different types of network structures that are missing in the original causal network.

A hypothesis of the form (2) can be obtained by adding a meta-level axiom:

$$connected(X, Y) \vee connected(X, Z) \leftarrow jointly\_connected(X, Y, Z). \quad (5)$$

to B and the atoms with the predicate $jointly\_connected(\_, \_, \_)$ to Γ.

Besides the use in rule abduction, meta-level abduction can also be applied to *fact abduction* [6], which has been focused on almost exclusively in AI literature.[2]

[2] In [25], fact abduction and rule abduction are classified as *factual abduction* and *law-abduction*, respectively. Our meta-level abduction also gives a realization of *2nd order existential abduction*, which is most important to produce new theories with new concepts [25].

Abduction of facts in the object level can be formalized as *query answering* in the meta level. Suppose that each abducible a in the object level is declared as $abd(a)$. Given a query of the form $\leftarrow caused(g, X)$, answer extraction for X can be realized by giving the clause of the form:

$$ans(X) \leftarrow caused(g, X) \wedge abd(X).$$

Here, ans is the *answer predicate* [9] and the variable X is used to collect abducibles which cause the g. Furthermore, by combining rule abduction and fact abduction in the form of *conditional query answering* [9], which extracts answers in a query with additional abduced conditions, meta-level abduction enables us to *abduce both rules and facts*.

All types of meta-level inferences in this section, including generation of existentially quantified hypotheses in meta-level abduction as well as conditional query answering to abduce rules and facts, can be realized by SOLAR [14,15]. SOLAR is a *consequence-finding* system based on SOL resolution [3] and the connection tableaux.

In SOLAR, the notion of production fields [3] is used to represent language biases for hypotheses. A *production field* $\mathcal{P}$ is a pair $\langle \mathbf{L}, Cond \rangle$, where $\mathbf{L}$ is a set of literals and $Cond$ is a certain condition. A clause C *belongs to* $\mathcal{P} = \langle \mathbf{L}, Cond \rangle$ if every literal in C is an instance of a literal in $\mathbf{L}$ and C satisfies $Cond$. The set of subsumption-minimal clauses derived from a clausal theory Σ and a production field $\mathcal{P}$ is called the *characteristic clauses* of Σ with respect to $\mathcal{P}$, and is denoted as $Carc(\Sigma, \mathcal{P})$. The *new characteristic clauses* of a clause C with respect to Σ and $\mathcal{P}$ are defined as $Newcarc(\Sigma, C, \mathcal{P}) = Carc(\Sigma \cup \{C\}, \mathcal{P}) \setminus Carc(\Sigma, \mathcal{P})$.

Given the background clausal theory B and the observations O, each abductive explanation H of O can be computed by *inverse entailment* [3]: $B \cup \{\neg O\} \models \neg H$, where $\neg O = \bigvee_{L \in O} \neg L$ and $\neg H = \bigvee_{L \in H} \neg L$ are clauses because O and H are sets of literals. Similarly, the condition that $B \cup H$ is consistent is equivalent to $B \not\models \neg H$. Given the abducibles Γ, any literal in $\neg H$ is an instance of a literal in $\overline{\Gamma} = \{\neg L \mid L \in \Gamma\}$. Hence, the set of minimal explanations of O with respect to B and Γ is characterized as $\{\, H \mid \neg H \in Newcarc(B, \neg O, \langle \overline{\Gamma} \rangle) \,\}$.

3 Reasoning about Positive and Negative Causal Effects

So far, links in a causal network have been of one kind, and $connected(g, s)$ in the meta level, i.e., $(g \leftarrow s)$ in the object level, just represents that g directly depends on s somehow. However, mixing different types of causalities in one type of links often makes analysis of actual causes complicated [17]. Here, we solve one of the most important problems of this kind: diagrams with two types of causalities, i.e., *positive* and *negative* causal effects. With this regard, from now on we can understand that each arc of the form $connected(g, s)$ in Section 2 only represents positive effects.

We extend applicability of meta-level abduction to deal with networks expressing both positive and negative causal effects. Such networks are seen in biological domains, where *inhibition* effects negatively in gene regulatory, signaling and metabolic pathways. Now we consider two types of direct causal relations: $triggered$ and $inhibited$. For two nodes g and t, the relation $triggered(g, t)$ represents a positive cause such that

t is a *trigger* of g, written as $g \longleftarrow t$ in a causal network, whose meaning is $(g \Leftarrow t)$ in the object level, where $\Leftarrow$ represents that the causation $\leftarrow$ appears if it is not prevented and its precise meaning will be defined later in Section 3.2. On the other hand, the relation $inhibited(g, s)$ represents a negative cause such that s is an *inhibitor* of g, written as $g \longmapsfrom s$ in a causal network, whose meaning is $(\neg g \Leftarrow s)$ in the object level.

As in Section 2, negation, disjunctive effects and conjunctive causes can be defined for $triggered$ and $inhibited$, cf., (1) and (2), and complex causal relations can be represented using those combinations and intermediate complexes. For instance, g is jointly triggered by t_1 and t_2 can be expressed as $triggered(g, t_1) \vee triggered(g, t_2)$. The notion of causal chains is also divided into two types: the positive one (written $promoted$) and the negative one (written $suppressed$), respectively corresponding to $triggered$ and $inhibited$. Now our task is to design the axioms for these two meta-predicates.

3.1 Alternating Axioms for Causality

Suppose first that there is no inhibitor in a causal network, that is, all links are positive. In this case, the axioms for $promoted$ should coincide with (3) and (4):

$$promoted(X, Y) \leftarrow triggered(X, Y). \tag{6}$$

$$promoted(X, Y) \leftarrow triggered(X, Z) \wedge promoted(Z, Y). \tag{7}$$

Next, let us interpret the meaning of an inhibitor as a toggle switch of signals flowed in the inhibitor, just as an inverter in a logic circuit [26]. Then, in the presence of inhibitors, we need one more axiom which blocks an adjacent inhibitor for X in order to promote X:

$$promoted(X, Y) \leftarrow inhibited(X, Z) \wedge suppressed(Z, Y). \tag{8}$$

As for the axioms of the negative causal chain $suppressed$, we can consider the following axioms, which are the counterpart of positive ones (6), (7) and (8):

$$suppressed(X, Y) \leftarrow inhibited(X, Y). \tag{9}$$

$$suppressed(X, Y) \leftarrow inhibited(X, Z) \wedge promoted(Z, Y). \tag{10}$$

$$suppressed(X, Y) \leftarrow triggered(X, Z) \wedge suppressed(Z, Y). \tag{11}$$

That is, a negative causal chain to X can be established if negative influence is propagated to X either directly by an adjacent inhibitor (9–10) or indirectly by (11).

One nice property with the axiomatization by (6–11) is that all possible paths from a source to a goal, which is either positive or negative, can be obtained by meta-level abduction. Meta-level abduction on causal networks with positive and negative links can now be defined by letting the abducibles Γ be those atoms with the predicates $triggered$ and/or $inhibited$: $\Gamma = \{triggered(\_, \_),\ inhibited(\_, \_)\}$, and observations or goals are given as literals either of the form $promoted(g, s)$ or of the form $suppressed(g, s)$. Hence, given positive and negative observations, we can abduce both positive and negative causes, and new nodes are produced whenever necessary.

Proposition 3.1. (Completeness) *If there is a negative (resp. positive) causal chain from a source s to a goal g ($g \neq s$) in a causal network N, then there is an explanation E of $suppressed(s, g)$ (resp. $promoted(s, g)$) from $N \cup \{(6–11)\}$ and Γ such that there exist an odd (resp. even) number of direct inhibitors in E.*

Conversely, it can be shown that the axiomatization (6–11) is *sound*: if there is an explanation of $\mathit{suppressed}(s, g)$ (resp. $\mathit{promoted}(s, g)$) from $N \cup \{(6\text{–}11)\}$, then there is a negative (resp. positive) causal chain from a source s to a goal g in the causal network N. However, this axiomatization can be *inconsistent* in the sense that both $\mathit{promoted}(g, s)$ and $\mathit{suppressed}(g, s)$ can be explained at the same time. This inconsistency is, however, inevitable in this monotonic representation since we can answer to any virtual query supposing a source s and a goal g. Then, to prevent derivations of promotion and suppression simultaneously for the same s and g, the following integrity constraint can be placed at the meta level.

$$\leftarrow \mathit{promoted}(X, Y) \wedge \mathit{suppressed}(X, Y). \tag{12}$$

The role of (12) is to derive *nogoods*, i.e., minimal incompatible combinations of instances of abducibles. Unfortunately, introduction of (12) can make those axioms inconsistent. For instance, it is easy to see that, for

$$N_0 = \{\mathit{triggered}(g, t), \mathit{triggered}(s, t), \mathit{inhibited}(g, s)\}, \tag{13}$$

$N_0 \cup \{(6\text{–}11), (12)\}$ is inconsistent. Also, the p53 network (23) given in Section 4 becomes inconsistent if it is combined with this axiomatization.

The problem arises when two antagonistic direct causal relations appear simultaneously for the same node g as follows.

$$\begin{tikzcd} & t \arrow[dl] & \mathit{triggered}(g, t) \\ g & & \\ & s \arrow[ul, -|] & \mathit{inhibited}(g, s) \end{tikzcd} \tag{14}$$

Our intuition on the diagram (14) is as follows. (1) If the trigger t is present and the inhibitor s is not present, then g is triggered by t; (2) Else if s is present and t is not present, then g is inhibited by s. These two cases are rather clear, but what happens for g if both t and s are somehow caused? The biological literature in this case indicates that: (3) If both t and s are present, then g is inhibited by s. Namely, *an inhibitor is preferred to a trigger*. The last inference is *nonmonotonic*: a trigger of g works *if there is no inhibitor for* g, but if an inhibitor is added then the trigger stops. We next show another axiomatization which reflects this principle of inhibitor preference.

3.2 Axiomatization with Default Assumptions

We now depart from the monotonic axiomatization of causal chains (6–11) to make reasoning about networks that are nonmonotonic. In the following new definitions of $\mathit{promoted}$ and $\mathit{suppressed}$, we will associate an extra condition for each trigger to work.

$$\mathit{promoted}(X, Y) \leftarrow \mathit{triggered}(X, Y) \wedge \mathit{no\_inhibitor}(X). \tag{15}$$

$$\mathit{promoted}(X, Y) \leftarrow \mathit{triggered}(X, Z) \wedge \mathit{no\_inhibitor}(X) \wedge \mathit{promoted}(Z, Y). \tag{16}$$

$$\mathit{promoted}(X, Y) \leftarrow \mathit{inhibited}(X, Z) \wedge \mathit{suppressed}(Z, Y). \tag{17}$$

$$\mathit{suppressed}(X, Y) \leftarrow \mathit{inhibited}(X, Y). \tag{18}$$

$$\mathit{suppressed}(X, Y) \leftarrow \mathit{inhibited}(X, Z) \wedge \mathit{promoted}(Z, Y). \tag{19}$$

$$\mathit{suppressed}(X, Y) \leftarrow \mathit{triggered}(X, Z) \wedge \mathit{no\_inhibitor}(X) \wedge \mathit{suppressed}(Z, Y). \tag{20}$$

$$\leftarrow \mathit{promoted}(X, Y) \wedge \mathit{suppressed}(X, Y). \tag{21}$$

The new axiom set for positive and negative causal chains (15–20) are the same as the monotonic version (6–11), except that each trigger to X ($\mathit{triggered}(X, \_)$) must not be inhibited ($\mathit{no\_inhibitor}(X)$) to give the positive effect to X in (15), (16) and (20). Here, inclusion of $\mathit{no\_inhibitor}(\chi)$ in association with $\mathit{triggered}(\chi, \psi)$ makes those three axioms *default rules*: the meaning $(\chi \Leftarrow \psi)$ in the object level is now given that the causation $\gamma = (\chi \leftarrow \psi)$ is true if γ is consistent with the union of the background theory B and a constructing hypothesis H. The literal of the form $\mathit{no\_inhibitor}(\_)$ is thus treated as a *default*, which can be assumed during inference unless contradiction occurs, and constraints can also be added to reject inconsistent cases with these assumptions. Finally, the integrity constraint (21) is the same as (12) and prohibits the presence of both positive and negative causes between any pair of nodes X and Y.

Meta-level abduction is now defined in the same way as in Section 3.1. Abduction of *joint triggers* or *joint inhibitors* can also be realized in the same way as $\mathit{jointly\_connected}$ in (5) by adding the meta-level axioms

$$\mathit{triggered}(X, Y) \lor \mathit{triggered}(X, Z) \leftarrow \mathit{jointly\_triggered}(X, Y, Z).$$
$$\mathit{inhibited}(X, Y) \lor \mathit{inhibited}(X, Z) \leftarrow \mathit{jointly\_inhibited}(X, Y, Z).$$

to the background theory B and the literals of the form $\mathit{jointly\_triggered}(\_, \_, \_)$ and of the form $\mathit{jointly\_inhibited}(\_, \_, \_))$ to the abducibles Γ. However, we do not need both of them if an intermediate complex is created; abduction of $\mathit{jointly\_inhibited}(g, s, t)$ can be simulated by abduction of $\mathit{jointly\_triggered}(s\text{-}t, s, t) \land \mathit{inhibited}(g, s\text{-}t)$.

As for default assumptions of the form $\mathit{no\_inhibitor}(\_)$, default reasoning can be implemented by assuming those literals whenever necessary during inference, and consistency of such assumptions are checked each time they are added to the current set of abduced literals. This is a simple yet powerful method for default reasoning in the case of so-called *normal defaults* [23,18]. Hence, the abducibles Γ now also contain the literals of the form $\mathit{no\_inhibitor}(\_)$, and are defined as

$$\Gamma = \{\, \mathit{triggered}(\_, \_),\ \mathit{inhibited}(\_, \_),\ \mathit{jointly\_triggered}(\_, \_, \_),\ \mathit{no\_inhibitor}(\_)\}.$$

When we are sure that there is no inhibitor for a node t, we can include the fact $\mathit{no\_inhibitor}(t)$ in the background theory B. For instance, $\mathit{no\_inhibitor}(s)$ can be declared as a fact in B if s is a terminal source node. Moreover, we can add a meta-level constraint

$$\leftarrow \mathit{no\_inhibitor}(X) \land \mathit{inhibited}(X, Y). \qquad (22)$$

This constraint (22) blocks to assume a default $\mathit{no\_inhibitor}(g)$ for any node g to which an inhibitor is connected.

Consistency of the background theory in the meta level containing a causal network, the new axioms (15–20) and constraint (21) is now always guaranteed. This is because, unlike the axioms (6–11), the new axioms contain additional defaults of the form $\mathit{no\_inhibitor}(\_)$. For the causal network N_0 (13), $B_0 = N_0 \cup \{(15\text{–}20), (21)\}$ is now consistent. Still, both $\mathit{promoted}(g, t)$ and $\mathit{suppressed}(g, t)$ can be explained from B_0 and Γ, but their explanations are not the same: $\{\mathit{no\_inhibitor}(g)\}$ explains the former, while $\{\mathit{no\_inhibitor}(s)\}$ explains the latter. Again, the role of (21) is

Table 1. Correspondence between object-level inference and meta-level consequence finding

object-level inference	top clause in SOLAR	production field in SOLAR
rule verification	$\leftarrow caused(g, s).$	$\langle \emptyset \rangle$
fact abduction	$ans(X) \leftarrow caused(g, X).$	$\langle \{ans(\_)\} \rangle$
fact prediction	$ans(X) \leftarrow caused(X, s).$	$\langle \{ans(\_)\} \rangle$
rule prediction	none	$\langle \{promoted(\_, \_), suppressed(\_, \_)\} \rangle$
rule abduction	$\leftarrow caused(g, s).$	$\langle \{\neg triggered(\_, \_), \neg inhibited(\_, \_)\} \rangle$
abducing rules and facts	$ans(X) \leftarrow caused(g, X).$	$\langle \{\neg triggered(\_, \_), \neg inhibited(\_, \_), ans(\_)\} \rangle$
fact prediction + rule abduction	$ans(X) \leftarrow caused(X, s).$	$\langle \{\neg triggered(\_, \_), \neg inhibited(\_, \_), ans(\_)\} \rangle$
rule prediction + rule abduction	none	$\langle \{\neg triggered(\_, \_), \neg inhibited(\_, \_), promoted(\_, \_), suppressed(\_, \_)\} \rangle$

to identify each nogood to prune all incompatible combinations of defaults and abducibles. That is, abducing literals with the predicates $triggered$ and $inhibited$ involves default assumptions of the form $no\_inhibitor(\_)$, and any inconsistent set of abducibles can be detected by subsumption checking with nogoods. In the network N_0, the set $\{no\_inhibitor(g), no\_inhibitor(s)\}$ is a nogood.

In abduction, completeness is guaranteed as in Proposition 3.1. This is easily proved as each explanation obtained with the monotonic axioms (6–11) can be extended by incorporating defaults of the form $no\_inhibitor(t)$ in the corresponding explanation from the axioms (15–20). Soundness of abductive explanations is similarly guaranteed.

Abduction with default assumptions has been implemented in SOLAR [7]. Membership of a clause C in an *extension* of a default theory [23,18] is guaranteed for each consequence

$$C \leftarrow no\_inhibitor(t_1) \wedge \cdots \wedge no\_inhibitor(t_m)$$

if $\{no\_inhibitor(t_1), \ldots, no\_inhibitor(t_m)\}$ is not a nogood.

Abduction of rules with positive and negative effects can be further combined with fact abduction to allow mixed forms of inferences. Table 1 summarizes the correspondence between object-level and meta-level inferences. All types of meta-level inferences, involving generation of existentially quantified hypotheses, can be realized by SOLAR. Recall that, in the context of inverse entailment, the negation of an observation is set to a *top clause*, and the negation of each abducible is given in a production field in SOLAR. In Table 1, "$\leftarrow caused(\_, \_)$" in a "top clause" column is instantiated by either "$\leftarrow promoted(\_, \_)$" or "$\leftarrow suppressed(\_, \_)$", and "$ans(\_)$" is an answer predicate to collect answer substitutions. In abducing object-level facts, a top clause can be further conditioned with an abducible literal "$abd(X)$" if the list of abducibles is given in the background theory B. In Table 1, "Rule verification" is to prove if a given causal chain can be derived or not. "Fact prediction" is to compute *ramification* of a source s, i.e., to derive facts that can be caused by s. "Rule prediction" is to enumerate possible causal chains from the given causal network, so a top clause is not provided in this case and characteristic clauses are computed by SOLAR.

4 Case Study: p53 Signal Networks

In this section, we see that meta-level abduction can be well applied to completion of *signaling networks*. The importance of network completion in signaling networks has been recognized since it is hard to observe activity levels and quantities of proteins in living organisms [1]. Moreover, reporter proteins/genes are usually employed in signaling pathways, but designing and introducing reporter proteins are hard tasks. This is contrasted to the case of genetic networks, in which expression levels of most genes can be observed using DNA microarray/chip technologies.

As case studies of meta-level abduction, we use two signaling networks, both of which contain the *p53 protein* [19] but use it for different purposes. Although these networks are rather simple, they illustrate one of the most fundamental inference problems in Systems Biology: Given an incomplete causal network, infer possible connections to promote or suppress some functions of biological systems. Those target functions are suppression of tumors in cancer and switching DNA synthesis on and off.

4.1 Enumerating Tumor Suppressors

This subsection examines the p53 signal network presented in [28] by meta-level abduction. The p53 protein plays a central role as a tumor suppressor and is subjected to tight control through a complex mechanism involving several proteins [19].

The p53 protein has the transactivator domain, which bounds to the promoters of target genes, then leads to protect the cell from cancer. The level and activity of p53 in the cell is influenced by its interactions with other proteins. Tumor suppression is enabled if the interacting partners of p53 do not inhibit the functionality of the transactivator domain. Mdm2 binds to the transactivator domain of p53, thus inhibiting the p53 from tumor suppression. UV (ultraviolet light) causes stress, which may induce the upregulation of p53. However, stress can also influence the growth of tumors.

These relations can be represented in solid lines of the causal network in Fig. 1. The corresponding formulas in the meta level can be simply represented by the clauses:

$$\begin{gathered} \mathit{triggered}(\mathtt{cancer}, \mathtt{uv}), \quad \mathit{triggered}(\mathtt{p53}, \mathtt{uv}), \\ \mathit{inhibited}(\mathtt{cancer}, a), \quad \mathit{triggered}(a, \mathtt{p53}), \\ \mathit{inhibited}(a, b), \quad \mathit{triggered}(b, \mathtt{p53}) \vee \mathit{triggered}(b, \mathtt{mdm2}), \end{gathered} \tag{23}$$

where a ("A" in Fig. 1) is the inhibitory domain of p53, and b ("B" in Fig. 1) is the complex p53-mdm2.

Now, we consider a tumor suppressor gene X such that mutants of X are highly susceptible to cancer. Suppose in some experiments that exposure of the cell to high level UV does not lead to cancer, given that the initial concentration of Mdm2 is high. A high level of gene expression of the X protein is also observed. Those initial conditions are represented as two facts, $\mathit{source}(\mathtt{uv})$ and $\mathit{source}(\mathtt{mdm2})$, that is, both UV and Mdm2 can be abduced whenever necessary. The meta-predicate source thus behaves like the abducible predicate abd. Some meta-level axioms can be introduced, e.g., $\mathit{no\_inhibitor}(X) \leftarrow \mathit{source}(X)$. Our objective is to hypothesize about the various possible influences of X on the p53 pathway thereby explaining how the cell can avoid cancer.

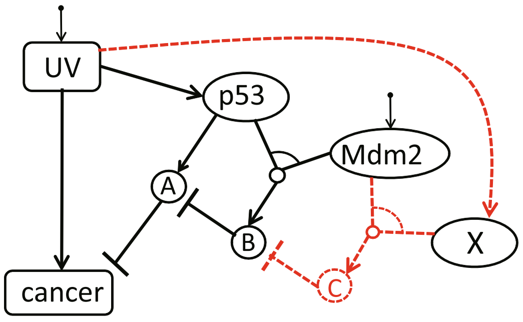


Fig. 1. Causal network of the p53 pathway

The observation is now expressed as $\exists S(\mathit{suppressed}(\mathtt{cancer}, S) \wedge \mathit{source}(S))$. Let

$$\Gamma = \{\ \mathit{triggered}(\_,\_),\ \mathit{inhibited}(\_,\_),\ \mathit{jointly\_triggered}(\_,\_,\mathtt{x})\ \},$$

be the abducibles, expecting that a mutant of X bound to some Z is produced in suppressing the cancer from some source. The background theory B is defined as the set consisting of the rules above, the causal axioms (15–20) and constraints (21,22), and domain constraints for pruning such as $\neg\mathit{inhibited}(\mathtt{uv}, Z)$ and $\neg\mathit{inhibited}(\mathtt{mdm2}, Z)$. In SOLAR, as in Table 1, the top clause is given as:

$$\mathit{ans}(S) \leftarrow \mathit{suppressed}(\mathtt{cancer}, S) \wedge \mathit{source}(S).$$

The production field $\mathcal{P}$ is set as: $\mathcal{P} = \langle\, \overline{\Gamma} \cup \{\mathit{ans}(\_), \mathit{no\_inhibitor}(\_)\},\ \mathit{Cond}\,\rangle$, where Cond is the length conditions such that each number of literals of the form $\neg\mathit{triggered}(\_,\_)$, $\neg\mathit{inhibited}(\_,\_)$ and $\neg\mathit{jointly\_triggered}(\_,\_,\mathtt{x})$ must not respectively exceed 1 in each produced clause. Then, SOLAR produces the 24 new characteristic clauses in 8 seconds using a PC with Core 2 Duo 3GHz and 4GB RAM.

In these 24 consequences of SOLAR, the following two clauses are included:

$$\begin{aligned} \mathit{ans}(\mathtt{uv}) \leftarrow\ & \mathit{triggered}(\mathtt{x}, \mathtt{uv}) \wedge \mathit{jointly\_triggered}(Y, \mathtt{p53}, \mathtt{x}) \\ & \wedge \mathit{inhibited}(b, Y) \wedge \mathit{no\_inhibitor}(Y) \quad (24) \\ \mathit{ans}(\mathtt{uv}) \vee \mathit{ans}(\mathtt{mdm2}) \leftarrow\ & \mathit{triggered}(\mathtt{x}, \mathtt{uv}) \wedge \mathit{jointly\_triggered}(Y, \mathtt{mdm2}, \mathtt{x}) \\ & \wedge \mathit{inhibited}(b, Y) \wedge \mathit{no\_inhibitor}(Y) \quad (25) \end{aligned}$$

Both (24) and (25) give conditional answers. The consequence (24) represents a *definite answer* indicating that the p53-X complex has the unique source UV since both p53 and X are caused by the same source UV. On the other hand, (25) represents a *disjunctive answer*: X is activated by UV but Mdm2 itself is assumed to be a source, hence the Mdm2-X complex has two sources. In fact, it takes more time to find the consequence (25) than to find (24) in SOLAR. Those two formulas respectively correspond to the following hypotheses:

(I) $\mathit{triggered}(\mathtt{x}, \mathtt{uv}) \wedge \exists Y(\mathit{jointly\_triggered}(Y, \mathtt{p53}, \mathtt{x}) \wedge \mathit{inhibited}(b, Y))$,
(II) $\mathit{triggered}(\mathtt{x}, \mathtt{uv}) \wedge \exists Y(\mathit{jointly\_triggered}(Y, \mathtt{mdm2}, \mathtt{x}) \wedge \mathit{inhibited}(b, Y))$.

The variable Y in (I) or (II) represents a new complex synthesized from X and either p53 or Mdm2, respectively. Those two hypotheses are actually suggested in [28]: (I) X directly influences p53 protein stability: Stress caused by UV induces high expression of X, which then binds to p53, so p53 is stabilized and formation of Mdm2-p53 complex is prevented; (II) X is a negative regulator of Mdm2: Stress induces high expression of X, which then binds to Mdm2, which competes against inhibiting the Mdm2-p53 interaction (depicted in dashed lines in Fig. 1). In both cases, p53 (or "A") can be functional as a tumor suppressor. In the biological viewpoint, however, the hypothesis (I) seems preferred because p53 has more chances to be bound to other proteins.

The new node Y in (I) and (II) is automatically invented in our method, while all ground candidate nodes and links to be added must be prepared as abducibles in [28].

The p53 regulatory network includes a complex array of upstream regulators and downstream effectors. The results obtained in this section are important in the sense that the activation/inhibition mechanism of p53 is linked to some proteins that might not have been found out yet. Meta-level abduction is thus crucial for this discovery task, and inferred hypotheses can suggest scientists necessary experiments with gene knockout mice as minimally as possible.

4.2 Recovering Links in CDK Networks

The next case study is completion in the switch network of *cyclin-dependent kinases* (CDKs) [26]. CDKs are kinase enzymes that modify other proteins by chemically adding phosphate groups to them (phosphorylation), and are involved in the regulation of the cell cycle. A CDK is activated by association with a cyclin, forming a cyclin-CDK complex (Fig. 2). The Cdk2/cyclin_E complex inactivates the retinoblastoma (Rb) protein, which then releases cells into the synthesis phase. Cdk2/cyclin_E is regulated by both the positive switch called CAK (cdk activating kinase) and the negative switch p21/WAF1. p21/WAF1 is activated by p53, but p53 can also inhibit cyclin_H, which is a source of the positive regulator of cdk2/cyclin_E. The negative regulation from p53 works as a defensive system in the cells: when DNA damage occurs, it triggers p53, which then turns on the negative regulation to stop DNA synthesis, so that the damage should be repaired before DNA replication to prevent passing damaged genetic materials onto the next generation.

For the CDK network in Fig. 2, we have used meta-level abduction to infer missing links. Experimental problems are designed by removing some links from Fig. 2, and then verifying if those links can be recovered or not in explaining the observation $suppressed(\texttt{dna\_synthesis}, \texttt{dna\_damage})$. The objective of this experiment is to show how meta-level abduction can be well applied to complete missing links. Recovery of removed links is a good testbed for this purpose because the existing natural system can be considered as an ideal solution. Yet, looking at other hypotheses, we can notice that the same functions can be realized in different ways.

Table 2 shows experimental results. All experiments are done in the environment on a Mac Mini with Core 2 Duo 1.83GHz and 2GB RAM. The table shows 6 problems, each of which is given a network obtained by removing the links shown in the table. The "#H" columns shows the number of new characteristic clauses (minimal hypotheses) by SOLAR. The unit of "Time" is second. "Depth" is the maximal search depth

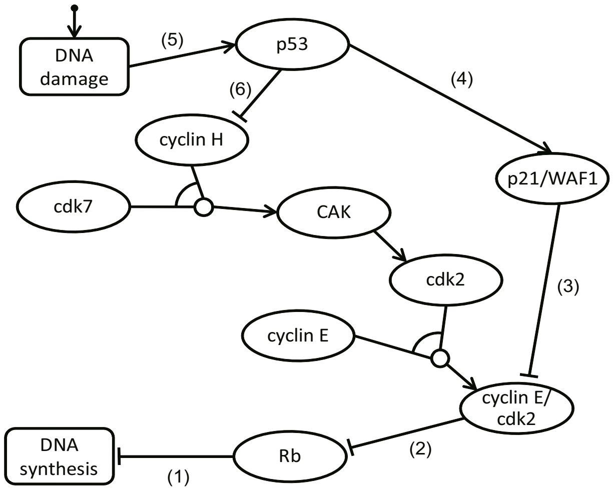


Fig. 2. Causal network of the CDK pathway

of SOLAR, and computation stops when no new consequences are derived at the minimum three successive depths k, $k+1$ and $k+2$, then we assume that all consequences have (probably) been found at depth k. The running time to obtain the set of new characteristic clauses is measured after the set of characteristic clauses (minimal nogoods) is computed. It takes 4 to 12 seconds to get all characteristic clauses.

The results of recovering removed links are shown in the bottom row. Here, the result indicates a hypothesis consisting of the links that are closest to the original links. For instance, when the links $\{(2),(4)\}$ are removed, then there exists the hypothesis containing exactly the same links as the removed ones in the 12 hypotheses. For a link (N), the recovered link (Ng) means that a more general hypothesis than (N) is obtained. For example, when $\{(1),(2)\}$, i.e., $\mathit{inhibited}(\texttt{dna\_synthesis}, \texttt{rb}) \land \mathit{inhibited}(\texttt{rb}, \texttt{cyclin\_e/cdk2})$ is removed, a more general hypothesis $\{(1\text{g}),(2\text{g})\}$, which is $\exists X(\mathit{inhibited}(\texttt{dna\_synthesis}, X) \land \mathit{inhibited}(X, \texttt{cyclin\_e/cdk2}))$, is recovered. We can observe that two consecutive links are replaced by general ones with existentially quantified variables at recovery, but links that are not connected are recovered as they are. Notice that only (4) is recovered when $\{(4),(6)\}$ is removed. This is because SOLAR outputs only minimal hypotheses, and only (4) is enough to explain the observation. Actually, the negative regulation by p53 has the two paths to suppress cdk2/cyclin_E via (4) and (6). Although this is a biologically robust system, our result indicates that only (4) is logically sufficient to realize this function.

5 Discussion and Related Work

This paper extends the method of *rule abduction* in [6] to deal with positive and negative causal links. Although few works on rule abduction exist previously, they focus on positive effects only unlike this paper. Poole [18] firstly considers *abducible rules*, which specify predefined patterns of rules for use in abduction. This is a very strong bias, and it is generally impossible to prepare all patterns of rules in advance and to perform

Table 2. Results of recovering links in the CDK pathway

Removed links	(1)		(1), (2)		(1), (3)		(2), (4)		(4), (6)		(1), (2), (3) (4), (5)	
Depth	#H	Time	#H	Time	#H	Time	#H	Time	#H	Time	#H	Time
3	1	0.6	1	0.6	1	0.7	1	0.7	1	0.7	1	1.1
4	2	0.6	3	0.6	3	0.7	4	0.7	5	0.8	4	1.5
5	3	0.7	4	1.3	5	1.2	8	1.3	6	1.2	12	1.9
6	3	1.3	4	1.7	6	1.6	11	1.9	8	1.8	25	2.8
7	4	**1.7**	5	**2.3**	7	**2.3**	12	**2.9**	8	**2.9**	37	**3.5**
8	*4*	2.2	*5*	3.2	*7*	3.5	*12*	3.9	*8*	5.0	*37*	5.7
9	*4*	2.5	*5*	3.8	*7*	4.2	*12*	6.0	*8*	5.5	*37*	6.9
Recovered links	(1)		(1g), (2g)		(1), (3)		(2), (4)		(4)		(1g), (2g), (3g) (4), (5)	

predicate invention. Work on Robot Scientist [10] adopts abduction to complete biochemical pathways without inhibition, and reaction edges in pathways are represented in a more complex manner [22]. Applications of SOLAR to complete metabolic pathways are also discussed in [20], but joint causes are not considered. CF-induction [4] can both abduce and induce rules, and its applications to complete causal rules as well as network status in metabolic pathways are shown in [29], but hypothesis enumeration and predicate invention are not easy tasks for CF-induction.

In ILP, some previous attempts contribute to *induction of causal rules* [11,16,5]. Moyle [11] requires the complete initial state as input and needs to compute a complete set of narrative facts in advance, and thus cannot account for observations handled in this paper. Otero [16] considers the case of incomplete narratives, but assumes that the truth value of a goal fluent changes only once in an incomplete narrative. Our algorithm, on the other hand, can induce any case in which the fluent value has changed more than once in intermediate situations. These previous works in ILP need either *frame axioms* or *inertia rules* in logic programs. The former causes the frame problem and the latter requires induction in nonmonotonic logic programs. Inoue *et al.* [5] uses an action language, but requires regular inference that searches the space of possible hypotheses.

There are several work on completing biological networks. *Metabolic pathways* are completed in [24] and are revised in [21] using answer set programming, although they do not invent new nodes. The work [24] does not consider inhibition. For revision [21], real deletion is generally impossible in biological networks unless they are subject to change. Hence, we only add new links and nodes by abduction, yet nonmonotonic features of inhibition are controlled by our method due to the axiomatization (15–20). Abduction in metabolic pathways with inhibition is considered in [27], although the problem in [27] is different from network completion and no new links/nodes are abduced. For *gene regulatory networks*, Gat-Viks and Shamir [2] determine a class of regulation functions, by which regulators determine transcription, and analyze their complexity. Zupan *et al.* [30] construct networks from mutant data using abduction, but use experts' heuristic rules for construction. Finally, completion of *signaling networks* is analyzed by Akutsu *et al.* [1], in which unknown Boolean functions are guessed in a Boolean network whose topology is known. This contrasts with our setting that a network is incomplete and its topology is not fixed. Tran and Baral [28] use an action language which formalizes notions such as *causal rules*, *trigger rules* and *inhibition rules* to model cell biochemistry, and apply it to hypothesize about signaling networks. As discussed in Section 4, all ground candidate nodes and links to be added are prepared as those abducible causal/trigger/inhibition rules in advance in [28].

6 Conclusion

The method of meta-level abduction [6] has been greatly extended in this paper to allow representation of positive and negative causal effects. With this extension, nonmonotonic reasoning in causal networks and abduction of positive and negative links are now possible. We have also shown an application to signaling pathways, and expect that the proposed method can be applied to abduction in other types of biological networks [2,27,29,21]. It is important to evaluate logically possible hypotheses, and some statistical methods can be applied. For example, hypotheses can be ranked according to frequencies of literals appearing in them and corresponding paths [8], and can be given their scores according to their fitness with observed data [2].

Problem solving with meta-level abduction consists of (i) design of meta-level axioms, (ii) representation of domain knowledge at the meta level, and (iii) restriction of the search space to treat large knowledge. This work supposes an incomplete network for the point (ii), whose representation is rather tractable. In fact, we have made great effort on the point (i), yet another axiomatization is possible for controlling inference in a different way. Possible extensions include introduction of time for causal chains and application of majority logic for determining values in competing cases. The point (iii) is achieved by introducing more constraints, and it is a future work to explore useful methods for inducing such constraints.

References

1. Akutsu, T., Tamura, T., Horimoto, K.: Completing Networks Using Observed Data. In: Gavaldà, R., Lugosi, G., Zeugmann, T., Zilles, S. (eds.) ALT 2009. LNCS, vol. 5809, pp. 126–140. Springer, Heidelberg (2009)
2. Gat-Viks, I., Shamir, R.: Chain functions and scoring functions in genetic networks. Bioinformatics 19(suppl.1), i108–i117 (2003)
3. Inoue, K.: Linear resolution for consequence finding. Artificial Intelligence 56, 301–353 (1992)
4. Inoue, K.: Induction as consequence finding. Machine Learning 55, 109–135 (2004)
5. Inoue, K., Bando, H., Nabeshima, H.: Inducing Causal Laws by Regular Inference. In: Kramer, S., Pfahringer, B. (eds.) ILP 2005. LNCS (LNAI), vol. 3625, pp. 154–171. Springer, Heidelberg (2005)
6. Inoue, K., Furukawa, K., Kobayashi, I., Nabeshima, H.: Discovering Rules by Meta-level Abduction. In: De Raedt, L. (ed.) ILP 2009. LNCS, vol. 5989, pp. 49–64. Springer, Heidelberg (2010)
7. Inoue, K., Iwanuma, K., Nabeshima, H.: Consequence finding and computing answers with defaults. Journal of Intelligent Information Systems 26, 41–58 (2006)
8. Inoue, K., Sato, T., Ishihata, M., Kameya, Y., Nabeshima, H.: Evaluating abductive hypotheses using an EM algorithm on BDDs. In: Proceedings of IJCAI 2009, pp. 810–815 (2009)
9. Iwanuma, K., Inoue, K.: Minimal Answer Computation and SOL. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS (LNAI), vol. 2424, pp. 245–257. Springer, Heidelberg (2002)

10. King, R.D., Whelan, K.E., Jones, F.M., Reiser, P.G.K., Bryant, C.H., Muggleton, S.H., Kell, D.B., Oliver, S.G.: Functional genomic hypothesis generation and experimentation by a robot scientist. Nature 427, 247–252 (2004)
11. Moyle, S.: Using Theory Completion to Learn a Robot Navigation Control Program. In: Matwin, S., Sammut, C. (eds.) ILP 2002. LNCS (LNAI), vol. 2583, pp. 182–197. Springer, Heidelberg (2003)
12. Muggleton, S.H., Bryant, C.H.: Theory Completion Using Inverse Entailment. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 130–146. Springer, Heidelberg (2000)
13. Muggleton, S., Buntine, W.: Machine invention of first-order predicate by inverting resolution. In: Proceedings of the 5th International Workshop on Machine Learning, pp. 339–351. Morgan Kaufmann, San Francisco (1988)
14. Nabeshima, H., Iwanuma, K., Inoue, K.: SOLAR: A consequence finding system for advanced reasoning. In: Cialdea Mayer, M., Pirri, F. (eds.) TABLEAUX 2003. LNCS, vol. 2796, pp. 257–263. Springer, Heidelberg (2003)
15. Nabeshima, H., Iwanuma, K., Inoue, K., Ray, O.: SOLAR: An automated deduction system for consequence finding. AI Communications 23(2–3), 183–203 (2010)
16. Otero, R.P.: Induction of the Indirect Effects of Actions by Monotonic Methods. In: Kramer, S., Pfahringer, B. (eds.) ILP 2005. LNCS (LNAI), vol. 3625, pp. 279–294. Springer, Heidelberg (2005)
17. Pearl, J.: Causality: Models, Reasoning, and Inference, 2nd edn., Cambridge (2009)
18. Poole, D.: A logical framework for default reasoning. Artificial Intelligence 36, 27–47 (1988)
19. Prives, C., Hall, P.A.: The p53 pathway. Journal of Pathology 187, 112–126 (1999)
20. Ray, O., Inoue, K.: A Consequence Finding Approach for Full Clausal Abduction. In: Corruble, V., Takeda, M., Suzuki, E. (eds.) DS 2007. LNCS (LNAI), vol. 4755, pp. 173–184. Springer, Heidelberg (2007)
21. Ray, O., Whelan, K., King, R.: Logic-based steady-state analysis and revision of metabolic networks with inhibition. In: Proceedings of the 3rd International Workshop on Intelligent Informatics in Biology and Medicine, pp. 661–666 (2010)
22. Reiser, P.G.K., King, R.D., Kell, D.B., Muggleton, S.H., Bryant, C.H., Oliver, S.G.: Developing a logical model of yeast metabolism. Electronic Transactions in Artificial Intelligence 5-B2(024), 223–244 (2001)
23. Reiter, R.: A logic for default reasoning. Artificial Intelligence 13, 81–132 (1980)
24. Schaub, T., Thiele, S.: Metabolic Network Expansion with Answer Set Programming. In: Hill, P.M., Warren, D.S. (eds.) ICLP 2009. LNCS, vol. 5649, pp. 312–326. Springer, Heidelberg (2009)
25. Schurz, G.: Patterns of abduction. Synthese 164(2), 201–234 (2008)
26. Shmulevich, I., Dougherty, E.R., Kim, S., Zhang, W.: Probabilistic Boolean networks: A rule-based uncertainty model for gene regulatory networks. Bioinformatics 18(2), 261–274 (2002)
27. Tamaddoni-Nezhad, A., Chaleil, R., Kakas, A., Muggleton, S.: Application of abductive ILP to learning metabolic network inhibition from temporal data. Machine Learning 65, 209–230 (2006)
28. Tran, N., Baral, C.: Hypothesizing about signaling networks. Journal of Applied Logic 7(3), 253–274 (2009)
29. Yamamoto, Y., Inoue, K., Doncescu, A.: Integrating abduction and induction in biological inference using CF-Induction. In: Lodhi, H., Muggleton, S. (eds.) Elements of Computational Systems Biology, pp. 213–234. John Wiley & Sons, Chichester (2010)
30. Zupan, B., Demsar, J., Bratko, I., Juvan, P., Halter, J.A., Kuspa, A., Shaulsky, G.: GenePath: A system for automated construction of genetic networks from mutant data. Bioinformatics 19(3), 383–389 (2003)

BET : An Inductive Logic Programming Workbench

Srihari Kalgi, Chirag Gosar, Prasad Gawde,
Ganesh Ramakrishnan, Kekin Gada,
Chander Iyer, Kiran T.V.S, and Ashwin Srinivasan

Department of Computer Science and Engineering
IIT Bombay, India
{cgosar,ganesh}@cse.iitb.ac.in
{chanderjayaraman,kekin.gada,prasg41,srihari.kalgi,t.kiran05}@gmail.com
ashwin.srinivasan@in.ibm.com
http://www.cse.iitb.ac.in/~bet

Abstract. Existing ILP (Inductive Logic Programming) systems are implemented in different languages namely C, Progol, etc. Also, each system has its customized format for the input data. This makes it very tedious and time consuming on the part of a user to utilize such a system for experimental purposes as it demands a thorough understanding of that system and its input specification. In the spirit of Weka [1], we present a relational learning workbench called BET(**B**ackground + **E**xamples = **T**heories), implemented in Java. The objective of BET is to shorten the learning curve of users (including novices) and to facilitate speedy development of new relational learning systems as well as quick integration of existing ILP systems. The standardized input format makes it easier to experiment with different relational learning algorithms on a common dataset.

Keywords: BET, ILP systems, Golem, Progol, FOIL, PRISM, TILDE.

1 Introduction

There have been several Inductive Logic Programming (ILP) system implementations. Each system has its own specifications for input data. Different systems do not necessarily agree on their inputs. This often makes comparisons across different implementations tricky, owing to either a difference in names or semantics of parameters and/or a difference in the choice of the programming language. Very often, the primitives and core components employed, such as theorem provers, SAT solvers, inference engines, etc., are also different across different implementations, rendering the computation and accuracy comparisons less reliable. This paper discusses a Workbench for ILP called BET. BET is developed in Java and it standardizes the specification of input using XML (eXtensible Markup Language). It provides a common framework for "building" as well as "integrating" different ILP systems. The provision for implementing algorithms in a common language (Java) improves the feasibility of comparing algorithms on their computational speeds. Whereas, the input standardization enables sound comparison of accuracies (or related measures) and also allows for experimenting with multiple ILP systems on the same dataset without any input conversion required to the system specific format.

BET includes several evaluation functions and operator primitives such as Least General Generalization (LGG) [9], Upward cover, Downward cover, etc. These features facilitate the rapid development of new relational learning systems and also ease out the integration of existing relational systems into BET. BET also allows a user to choose from multiple theorem provers as plug and play components. Presently, YAP (Yet Another Prolog) [12] and SWI-Prolog [13] are included in the basic package. The initial version of BET has three relational systems implemented namely FOIL [4], Golem [3] and TILDE [2][10] and four relational systems integrated namely Progol [11], FOIL, Golem and PRISM [5][6] (Though PRISM is not a standard ILP system, it has been integrated with the specific intension of learning probabilities in order to associate uncertainity with theories learnt using ILP systems).

We proceed with an overview of BET and the motivation to develop such a system in Section 2. Section 3 focuses on the design of the BET system. Section 4 explains how new relational learning systems can be integrated/implemented in BET. Section 5 compares BET with existing workbenches and highlights the advantages of BET. We summarize BET in Section 6.

2 Overview

A principal motivation for developing a system like BET is the reduction of the learning curve for both expert-users and novices as well as programmers in the area of relational learning, particularly ILP. For example, with BET, the end-user will need to understand only the standardized input parameters for BET. This reduces the time overheads involved in comparing different algorithmic implementations as well as in experimenting with individual implementations. More specifically, a user need not convert datasets from one format to another while switching between ILP systems. Further, the standardized APIs in BET make development of algorithms within BET much easier. Figure 1 shows the block diagram of BET.

Any ILP system in BET takes four files as its input namely positive examples, negative examples, background knowledge and language restriction files, and it outputs in the form of *theories*. There are primarily two important components of BET namely the *BET GUI* and the *BET Engine*. The BET Engine is the back-end engine at the core of BET and the BET GUI communicates with the BET engine through the medium of a *properties* file. BET users can experiment with the system using the GUI whereas programmers can use the APIs provided and extend BET with new ILP or relational learning systems. Section 5 explains how BET could be extended.

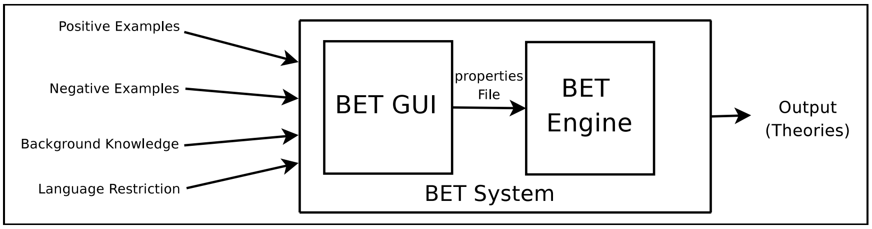


Fig. 1. Block diagram of BET

3 Design of BET

Almost all ILP algorithms as well as systems require following five inputs: positive examples, negative examples, background knowledge, mode declarations and hyper-parameters for tuning the system. The clauses learnt may be customized by means of the mode declarations, though some algorithms/systems may not allow for the specification of mode declarations (example is Confidence-based Concept Discovery system [7]). In this Section, we discuss the need for having a standard for the input, and eventually specify the BET standard input file formats.

3.1 Need for Standard Format

There is little or no consensus between the input specifications of ILP systems. For example, FOIL has its standard input specification which is completely different from that of Golem or Progol. FOIL accepts the positive examples, negative examples and background knowledge in a single file. On the other hand, Golem takes three files as input viz. positive example file ($.f$), negative example file ($.n$) and the background knowledge file ($.b$) which includes the mode declarations. Another ILP system Progol has an input format that is slightly similar to Golem, but has many more setting parameters.

Clearly any end user, desiring to experiment with a new ILP system, is required to understand its input format thoroughly, convert the data in hand to that format and only then can he/she use it. BET assists the end user with a standardized input specification that is generic enough and can be converted to the specification of any existing ILP system. Section 3.2 will discuss about BET standard formats.

3.2 BET Standard Format

As already stated, BET's inputs are comprised of four XML files namely positive examples, negative examples, background knowledge and language restriction file. Sample positive example, negative example, background knowledge and language restriction files for learning the *ancestor* relation are shown in *Table 1*, *Table 2*, and *Table 3* respectively in the document available at [15]. The hyper-parameters can be specified in Language Restriction file. The format for each file is as follows:

Positive Example File: The file consists of clauses which represents positive examples for the relation (predicate) for which the theory needs to be learned.

Negative Example File: This file consists of clauses which represents negative examples of the predicate (relation) for which the theory needs to be learned. There may be some ILP systems which do not consider any negative examples during construction of theory, but may utilize negative examples for pruning the theories that have been learned.

Background knowledge File: The background knowledge (BGK) file contains all the *domain specific information* which is required by the ILP system in order to learn the target predicate. All this information should be in the form of clauses. It can also contain rules instead of just facts.

Language Restriction File: The structured search space (lattice or simply a partial order) of clauses which needs to be searched in order to arrive at a reasonably good hypothesis is almost infinite. *Language Restrictions* endow an ILP system with the flexibility of reducing the search space. There are three generic parts to the Language Restriction viz. Mode declaration, Type and Determination. Also we can specify the system-specific hyper-parameters like stopping threshold, minimum accuracy, etc. in this file only. Each of them, is explained below.

Type: Type defines the type of constants. For example, the constant `john` is of type `person`, constant `1` is of type `integer`. Type specification can be done in the language restriction file.

Mode Declarations: Mode declarations declare the mode of the predicates that can be present in any of clausal theories. The mode declaration takes the form `mode(RecallNumber,PredicateMode)`, where
RecallNumber: bounds the non-determinacy of a form of predicate call.
PredicateMode: has the following syntax

```
      predicate(ModeType,ModeType,...)
    predicate(<+/-/#>Type,<+/-/#>Type,...)
```

+Type : the term must be an input variable of type Type.
-Type : the term must be an output variable of type Type.
#Type : the term must be a ground constant of type Type.

Determination: This component of mode declarations provides information regarding which predicates need to be learned in terms of which other predicates.

Hyper-parameters: Hyper-parameters are very much system specific and we can find the different parameters for a system by doing *system-name –help* or through the graphical interface for that particular system.

4 Class Diagram and Components of BET

The class diagram of BET is shown in Figure 1 in the document available at [15]. The major components of BET are explained in subsequent sections.

ClauseSet: ClauseSet refers to a set of clauses, which is often referred to in the description of ILP algorithms. The ClauseSet interface defines the signature of the methods used to access the ClauseSet object.

LanguageRestriction: LanguageRestriction is an interface for mode declarations, determinations and types.

InputEncoder: InputEncoder is mainly used for integrating legacy systems (i.e. ILP systems written in native code) into BET. Any wrapper around an integrated legacy system will require conversion of the BET standard data format to the format accepted by the legacy system. Any legacy system which requires to be integrated into BET has to implement `InputEncoder` interface and provide a method of converting the standard BET file format to the format accepted by the legacy system.

TheoryLearner: TheoryLearner is an important component of BET. It is responsible for learning the theory (hypothesis) from the input. `TheoryLearner` has objects of type `InputEncoder` and `EvalMetric` (explained next) as its data members. In case the ILP system is completely implemented in Java (i.e., if it is not a legacy system), it can use a dummy implementation of the `InputEncoder` interface and pass the object to the `TheoryLearner` class. The dummy implementation of `InputEncoder` interface is mainly used to access the BET input files.

EvalMetric (Evaluation Metric): Any ILP algorithm will try to explore the structured space (subsumption lattice or partial order) of clauses, by traversing the graph using refinement operators. At every point in the space, the algorithm has to decide whether a clause in the space is good enough to be present in the hypothesis. For this purpose, the covers relation is employed.

If the covers relation is defined by `Entailment` ($\models$), then the ILP system is said to learn from entailment [9]. Some other ILP systems learn from proof traces [9], while few others learn from interpretations [9], etc.

Any implementation of EvalMetric will specify the *covers* relation, *i.e.*, definition of an example being covered by background knowledge (with candidate hypothesis).

TheoremProver: The theorem prover is the backbone of most ILP systems. BET has two different theorem provers namely YAP and SWI-Prolog. YAP is perhaps the fastest theorem prover as of today. YAP is built in C and doesn't have any JNI (Java Native Interface), so the YAP process is run by spawning a process through a BET program. SWI-Prolog is also built in C, but it does support JNI (Java Native Interface). SWI-Prolog has a standard interface called JPL which comes bundled with SWI-Prolog itself.

5 Support for Building ILP Systems in BET

It is very easy to integrate[1] an existing system into BET. Also the implementation[2] of a new system within BET is faster than implementing it from scratch. The following section briefs on how to integrate and implement a system in BET.

5.1 Integrating a Legacy System in BET

Legacy systems can be integrated into BET as follows:

– Implement the interface `InputEncoder`, which should provide a functionality to convert files from BET standard format to the format accepted by the legacy system.
– Extend the `TheoryLearner` class and override the method `learnTheory`.

5.2 Implementing a New ILP System in BET

– A dummy implementation of the interface `InputEncoder` is required for this purpose, since the `TheoryLearner` class expects an object of `InputEncoder` while creating its object.
– Extend the `TheoryLearner` class and override the method `learnTheory`.
– Different types of Evaluation Metric, Theorem provers (YAP, SWI-Prolog) can be used in implementing the ILP system.

6 Related Work

Aleph and GILPS (Generic Inductive Logic Programming System) are two earlier efforts put in to develop a workbench like BET for Inductive Logic Programming. Aleph [8] is developed by Ashwin Srinivasan, whereas the GILPS [14] workbench is developed by Jose Santos.

6.1 Aleph

Aleph is written in Prolog principally for use with Prolog compilers like Yap and SWI-Prolog compiler. Aleph offers a number of parameters to experiment with its basic algorithm. These parameters can be used to choose from different search strategies, various evaluation functions, user-defined constraints and cost specifications. Aleph also allows user-defined refinement operators.

6.2 GILPS

GILPS implements three relational systems namely TopLog, FuncLog and ProGolem. GILPS requires at least YAP 6.0 for its execution. Many of its user predicates and settings are shared with Aleph.

[1] The authors of this paper took two weeks to integrate PRISM in BET and one week individually for integrating Golem and FOIL.

[2] The authors of this paper implemented TILDE, FOIL and Golem in BET within a week for each algorithm.

6.3 Advantages of BET over Existing Systems

BET offers following advantages over existing systems (Aleph, GILPS ,etc.):

- BET is more likely to find use in the larger applied machine learning and Mining communities who are less familiar with prolog and where it is important to interface learning tools in Java (such as BET) with applications largely written in procedural languages (very often Java). This can also make diagnosis and debugging easy. As against this, GILPS and Aleph assume familiarity with Prolog and cannot be easily invoked from applications written in Java.
- In the spirit of Weka, BET provides a nice framework for the integration of legacy systems as well as for the implementation of new systems. In fact, while the basic BET framework was provided by the first two co-authors, the remaining co-authors used that framework to implement several ILP algorithms. This is something that Aleph and GILPS do not cater to as well.
- BET allows saving the learned models as serialized files which can later be used for finding models for new problems.
- BET provides a choice of theorem prover where as GILPS can work only with YAP.
- BET comes with a YAP installer and all the "integrated" systems.
- BET has classes to convert the background knowledge, positive examples and negative examples files into XML format. Language restriction file needs little human intervention for this purpose.
- BET provides a standard input/output format which enables end-user to experiment with multiple relational systems with the same datasets. Aleph uses three different files namely (.f), (.n) and (.b) whereas GILPS takes input as prolog files (.pl).

7 Summing Up BET

The standardized input format of BET makes it more convenient for user to have only one input format for all relational systems. The ability of BET to write learned classification/regression models as serialized files allows for saving the model and reusing the model at a later stage for classification or regression on new problems. The BET graphical user interface makes experimentation with implemented algorithms and integrated systems much more easier. Use of JAVA as an implementation language makes BET extensible and platform independent. The only visible disadvantage of BET is that the JAVA implementation of relational systems are slower as compared to their C/C++/Progol counterparts, but we chose Java over C++ owing to ease of extending and platform independence of the former over the latter.

References

1. Frank, E., Hall, M.A., Holmes, G., Kirkby, R., Pfahringer, B., Witten, I.H.: Weka-A machine learning workbench for data mining. In: Data Mining and Knowledge Discovery Handbook- A Complete Guide for Practitioners and Researchers, pp. 1305–1314. Springer, Berlin (2005)
2. Blockeel, H., De Raedt, L.: Top-down induction of first-order logical decision trees. Artificial Intelligence Journal, 285–297 (1998)
3. Muggleton, S.H., Feng, C.: Efficient induction of logic programs. In: Proceedings of the First Conference on Algorithmic Learning Theory, Ohmsha, Tokyo, pp. 368–381 (1990)
4. Ross Quinlan, J., Mike Cameron-Jones, R.: FOIL: A Midterm Report. In: Brazdil, P.B. (ed.) ECML 1993. LNCS, vol. 667, pp. 3–20. Springer, Heidelberg (1993)
5. Sato, T., Kameya, Y., Zhou, N.-F.: Generative Modeling with Failure in PRISM. In: IJCAI, pp. 847–852 (2005)
6. Sato, T., Kameya, Y.: PRISM: A Language for Symbolic-Statistical Modeling. In: IJCAI, pp. 1330–1339 (1997)
7. Kavurucu, Y., Senkul, P., Toroslu, I.H.: ILP-based concept discovery in multi-relational data mining. Expert Systems with Applications 36, 11418–11428 (2009)
8. Aleph Manual, `http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/aleph.html`
9. Statistical Relational Learning Notes, `http://www.cse.iitb.ac.in/~cs717/notes/`
10. TILDE: Top-down Induction of Logical Decision Trees, `http://www-ai.ijs.si/~ilpnet2/systems/tilde.html`
11. Progol, `http://www.doc.ic.ac.uk/~shm/progol.html`
12. YAP Manual, `http://www.dcc.fc.up.pt/~vsc/Yap/`
13. SWI-Prolog, `http://www.swi-prolog.org/`
14. General Inductive Logic Programming System, `http://www.doc.ic.ac.uk/~jcs06/GILPS/`
15. ILP Systems Integrated Implemented in BET and Sample Examples, `http://www.cse.iitb.ac.in/~bet/appendix.pdf`

Seeing the World through Homomorphism: An Experimental Study on Reducibility of Examples

Ondřej Kuželka and Filip Železný

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

Abstract. We study reducibility of examples in several typical inductive logic programming benchmarks. The notion of reducibility that we use is related to theta-reduction, commonly used to reduce hypotheses in ILP. Whereas examples are usually not reducible on their own, they often become implicitly reducible when language for constructing hypotheses is fixed. We show that number of ground facts in a dataset can be almost halved for some real-world molecular datasets. Furthermore, we study the impact this has on a popular ILP system Aleph.

1 Introduction

In this paper, we are interested in identification of learning examples which are not distinguishable given a fixed hypothesis language. We propose a notion of *safe reduction* and show how it can be exploited in order to obtain smaller examples equivalent to the original ones.

2 Preliminaries

Let us first state some notational conventions used in this paper. The set of literals in a clause C is written as $lits(C)$, $|C| = |lits(C)|$ is the size of C. The set of variables in a clause C is written as $vars(C)$ and the set of all terms by $terms(C)$. A substitution θ is a mapping from variables of a clause C to terms of a clause D.

Let us now define θ-reduction [4], which will be later utilized for reduction of training examples. In order to define θ-reduction, we need the notion of θ-subsumption [5]. Let us denote the set of variables contained in a clause C by $vars(C)$. We say that a clause C θ-subsumes clause D (denoted by $C \preceq_\theta D$) if and only if there is a substitution $\theta : vars(C) \rightarrow vars(D)$ such that $C\theta \subseteq D$. It holds that θ-subsumption implies entailment, but not vice-versa, i.e. if $C \preceq_\theta D$ then $C \models D$, but if $C \models D$ then $C \preceq_\theta D$ may or may not be true.

Definition 1 (θ-Reduction). *Let C and D be clauses. We say that C and D are θ-equivalent (denoted by $C \approx_\theta D$) if and only if $C \preceq_\theta D$ and $D \preceq_\theta C$. If there is a clause E such that $C \approx_\theta E$ and $|E| < |C|$ then C is said to be θ-reducible.*

For example, $C = east(T) \leftarrow hasCar(T,C) \wedge hasLoad(C,L1) \wedge hasLoad(C,L2) \wedge box(L2)$ is θ-reducible because $C \approx_\theta E$ where $E = east(T) \leftarrow hasCar(T,C) \wedge hasLoad(C,L) \wedge box(L)$. In this short paper, we will work within the learning from entailment setting [1].

Definition 2 (Learning from Entailment). *Let H and e be clausal theories. Then we say that H covers e under entailment (denoted by $H \preceq_E e$) if and only if $H \models e$.*

For example, let $e = east(t1) \leftarrow hasCar(t1,c1) \wedge hasLoad(c1,l1) \wedge box(l1)$ and let $H = \forall T \forall C : east(T) \leftarrow hasCar(T,C)$. Then we may easily check that H covers e under entailment (i.e. $H \preceq_E e$). In what follows we will refrain from writing the universal quantifiers in clauses.

3 Reducing the Examples

The learning task that we study in this paper is fairly standard. We are given a set of positive and negative examples encoded as first-order clauses and we would like to find a classifier separating the positive examples from the negative examples. This task could be solved by numerous ILP systems. We aim at finding a reduction procedure that would allow us to reduce the number of atoms in the examples while guaranteeing that the coverage of individual hypotheses would not be changed. This is formalized by the next definition which introduces the concept of safe reduction under intepretations.

Definition 3 (Safe Reduction under Entailment). *Let e and $\hat{e}$ be two interpretations and let $\mathcal{L}$ be a language specifying all possible hypotheses. Then $\hat{e}$ is said to be a safe reduction of e under entailment if and only if $\forall H \in \mathcal{L} : (H \preceq_E e) \Leftrightarrow (H \preceq_E \hat{e})$ and $|\hat{e}| < |e|$.*

Clearly, any hypothesis H which splits the positive examples from the negative examples correctly will also split the respective safely reduced examples correctly. Also, when predicting classes of test-set examples, any deterministic classifier that bases its decisions on the queries using solely the covering relation $\preceq_E$ will return the same classification even if we replace some of the examples by their safe reductions[1]. On the other hand, even the classifiers constructed by a deterministic learner on reduced and non-reduced examples may be different. For example, ILP systems that restrict their search space by bottom clauses may return different hypotheses for reduced and non-reduced examples. However, if the search is performed exhaustively, every hypothesis discovered for the reduced data will have to cover the same set of examples as some corresponding hypothesis discovered for the non-reduced data.

Let us now look at possible candidates for safe reduction. For example, θ-reduction $\hat{e}$ of a ground example e is a safe reduction because trivially $\hat{e} = e$.

[1] This is also true for propositionalization approaches that use the $\preceq_E$ relation to construct boolean vectors which are then processed by attribute-value-learners.

Assuming that in practical learning tasks, examples are very often ground, θ-reduction alone does not seem to help us very much. It turns out that we need some additional assumptions in order to be able to reduce even ground examples. Before we present the (admittedly very simple) kind of safe reduction that will be in the center of our interest in this paper, we go through the next simple motivating example.

Example 1. Suppose that we have one positive example $e_1^+ = east(t1) \leftarrow hasCar(t1, c1) \wedge hasLoad(c1, l1) \wedge hasCar(t1, c2)$ and one negative example $e_2^- = east(t2) \leftarrow hasCar(t2, c3)$. Our task is to learn a non-recursive definition of predicate $east/1$. Let the language $\mathcal{L}$ contain all non-recursive Horn clauses free of functions and constants and containing only predicate $east/1$ in the head and predicates $hasCar/2$ and $hasLoad/2$ in the body. Because of the non-recursivity and function-freeness assumptions we can check whether $H \preceq_E e$ is true by testing θ-subsumption between the hypothesis and the example. Let e' be a clause obtained by variabilizing the clause e. Since constants are not allowed in the hypothesis language $\mathcal{L}$, it holds that $H \preceq_\theta e \Leftrightarrow H \preceq_\theta e'$. Next, let $\hat{e}'$ be θ-reduction of e'. It also holds $H \preceq_\theta e \Leftrightarrow H \preceq_\theta \hat{e}'$. Clearly, if $H \preceq_\theta e$, there is a substitution θ such that $H\theta \subseteq e$. If we replace the constants in θ by the respective variables used when variabilizing e and obtain a substitution θ', it will also hold $H\theta' \subseteq e'$. Finally, since there is a substitution θ_{R1} such that $e'\theta_{R1} \subseteq \hat{e}'$ (because $e' \approx \hat{e}'$), it must also hold $H \preceq_\theta \hat{e}'$ because $H\theta'\theta_{R1} \subseteq \hat{e}'$. Similarly, we could justify the implication in the other direction, i.e. $H \preceq_\theta e \Leftarrow H \preceq_\theta \hat{e}'$.

Applied to the example e_1^+, we have that $\hat{e}_1^+ = east(T1) \leftarrow hasCar(T1, C1) \wedge hasLoad(C1, L1)$ is a safe reduction of e_1^+ w.r.t. the language $\mathcal{L}$. Notice that it is important to have a fixed hypothesis language to perform this kind of reductions. For example, if we allowed the constants $c1$, $c2$ and $c3$, $\hat{e}_1^+$ would no longer be a safe reduction of e_1^+ because the hypothesis $H = east(T) \leftarrow hasCar(T, c1) \wedge hasCar(T, c2)$ would cover e_1^+ but not $\hat{e}_1^+$.

We are now ready to describe the reduction method. The method expects the examples encoded as first-order clauses and a description of the hypothesis language on its input. The hypothesis language specification should provide information about which predicates and constants can be used to build the hypotheses. It starts by variabilizing the constants which are not in the hypothesis language but appear in the examples. After that, examples are reduced using θ-reduction and these reductions are output by the algorithm. In order to justify correctness of this procedure, we need the next almost trivial proposition.

Proposition 1. *Let $\mathcal{L}$ be a hypothesis language and let e be a clause. (i) Let $\tilde{e}$ be a clause obtained from e by variabilizing the constants which are not contained in the hypothesis language. Then $(H \preceq_E e) \Leftrightarrow (H \preceq_E \tilde{e})$ for any $H \in \mathcal{L}$ (ii) Let $\hat{e}$ be θ-reduction of $\tilde{e}$. Then $(H \preceq_E e) \Leftrightarrow (H \preceq_E \hat{e})$ for any $H \in \mathcal{L}$.*

The first part of Proposition 1 justifies the step of the reduction, in which some constants are replaced by variables, and the second part justifies the step, in which the examples are reduced using θ-reduction. Optionally, we may perform also a third step in which the variables introduced in the first step are converted back to constants.

Example 2. Let us have a hypothesis language for the task of predicting mutagenicity of molecules $\mathcal{L}$. Let us have one example

$$\begin{gathered} e = pos(m) \leftarrow bond(a1, a2, 2) \wedge bond(a2, a1, 2) \wedge a(m, a1, c) \wedge a(m, a2, c) \\ \wedge bond(a1, a3, 1) \wedge bond(a3, a1, 1) \wedge a(m, a3, h) \wedge a(m, a4, h) \wedge bond(a2, a5, 1) \\ \wedge bond(a5, a2, 1) \wedge a(m, a5, h) \wedge bond(a2, a6, 1) \wedge bond(a6, a2, 1) \wedge a(m, a6, h). \end{gathered}$$

In the hypothesis language $\mathcal{L}$ we do not consider the names of atoms (i.e. constants $a1$, $a2$ etc). We start by performing the first step of the reduction procedure, i.e. by variabilizing the constants that do not appear in the hypothesis language $\mathcal{L}$. We replace constant $a1$ by variable $A1$, $a2$ by $A2$ and so on and obtain a clause $\tilde{e}$. After this step, we compute the θ-reduction of $\tilde{e}$, which is

$$\begin{gathered} \hat{e} = pos(m) \leftarrow bond(A1, A2, 2) \wedge bond(A2, A1, 2) \wedge atm(m, A1, c) \wedge \\ \wedge a(m, A2, c) \wedge bond(A1, A3, 1) \wedge bond(A3, A1, 1) \wedge \wedge a(m, A3, h) \wedge . \\ bond(A2, A5, 1) \wedge bond(A5, A2, 1) \wedge a(m, A5, h), \end{gathered}$$

The θ-reduction of $\hat{e}$ is witnessed by the substitution $\theta = \{A4/A3, A6/A5\}$. It is easy to check that $e\theta \subseteq \hat{e}$. Clause $\hat{e}$ is a safe reduction of e.

4 Experiments

In this section, we discuss reducibility of examples from real-life datasets using typical hypothesis languages. We show that, in the most extreme case, some examples are reduced to one tenth of their original length. Then we study the effect example reductions have on Aleph. The θ-reduction is performed by an algorithm built upon the θ-subsumption system ReSumEr2 [3]. Reduction runtimes were in all cases under 30 seconds, which is a negligible amount of time compared to time consumed by Aleph as we will see. In all of the experiments we deal only with ground examples and non-recursive clauses. Therefore, for these experiments, we do not need the whole logical formalism. It would suffice to consider only θ-subsumption (homomorphism).

4.1 Mutagenesis

The first set of experiments which we performed was done with the well-known Mutagenesis dataset [6]. In the encoding which we used and which is also usually used in the experiments with most ILP systems, every molecule is described using predicates $atm(M, A, T, T2)$ and $bond(A, B, BT)$. For example, let us have a literal $atm(mol, atom1, c, 22)$. Here, mol is an identifier of the molecule, $atom1$ is an identifier of the atom, the third argument denotes the atomic type, here c means carbon, and the constant 22 in the fourth argument means that $atom1$ is an *aromatic atom*. Similarly, $bond(atom1, atom2, 1)$ denotes a single bond

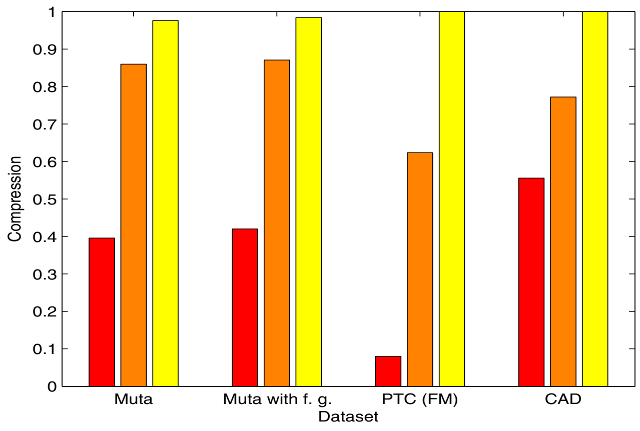


Fig. 1. Compression rates (*number of facts in reduced example / number of facts in original example*) obtained for three real-life datasets (Mutagenesis [6], Mutagenesis with function groups, PTC [2] and CAD [7]). The bars show maximum, average and minimum obtained compression.

between atoms $atom1$ and $atom2$. Each atomic bond is represented by two such $bond/3$ literals. In this case, when testing $H \preceq_E e$ with a single non-recursive clause H, the covering relation $\preceq_E$ corresponds to homomorphism testing (or θ-subsumption testing[2]). We let the hypothesis language $\mathcal{L}$ contain only constants corresponding to atomic types and numbers. After applying the safe reduction procedure, the number of literals in the examples decreased to 86% on average. The most succesful reduction decreased the number of literals in an example to 40% of the original count.

Next, we enriched the examples and the hypothesis language $\mathcal{L}$ by description of function groups such as benzene, phenathrene etc. This corresponds to another frequently used setting. Interestingly, the reduction rates differed only by a very small number from the previous case. The number of literals decreased to 87% on average and, for the most succesful reduction, the number of literals in one example decreased to 42% of the original count of literals.

4.2 Predictive Toxicology Challenge

The next experiment was performed with the Predictive Toxicology Challenge dataset (PTC). PTC dataset consists of molecules marked according to their carcinogenicity for female mice, male mice, female rats and male rats. Since the examples are almost the same for all of the four cases (with the exception of a few molecules contained in only some of the four datasets), we performed our experiments only for female mice. The atoms in this dataset are described at a cruder level compared to the Mutagenesis dataset. Only the basic types

[2] Checking homomorphism of labeled hypergraphs, checking θ-subsumption of clauses and solving constraint satisfaction problems is essentially the same thing.

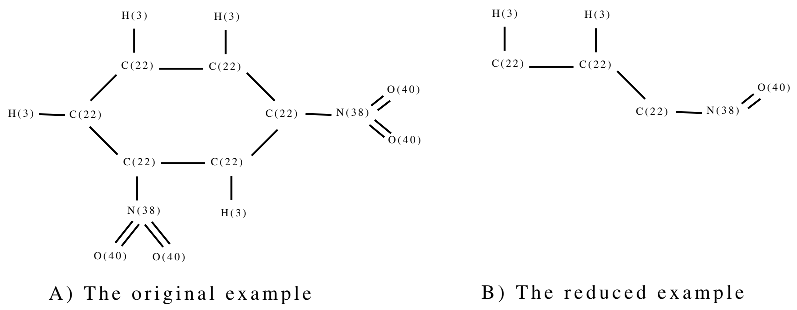


Fig. 2. The example from the Mutagenesis dataset which was compressed most by the reduction procedure. **Left:** The original example. **Right:** The reduced example. Note that every bond is represented by a pair of *bond*/3 literals.

like *carbon* and not *aromatic carbon* are used. A consequence of this is that higher reduction rates are possible. After applying the safe reduction procedure, the number of literals in the examples decreased to 62% on average. The most succesful reduction decreased the number of literals in an example to just **8%** of the original count.

4.3 CAD

The last experiment was performed with the CAD dataset [7]. This dataset contains 96 class-labeled product-structures desings. After applying the safe reduction procedure to the examples from the CAD dataset, the number of literals in them decreased to 77% on average. The most succesful reduction decreased the number of literals in an example to 56% of the original count. The reason for smaller difference between average and maximum reduction rates as compared to the previous two datasets was that the designs contain *next*/2 literals which express a sort of ordering of the elements, which limits applicability of the θ-reduction.

4.4 Aleph with Reduced and Non-reduced Data

Finally, we decided to compare performance of Aleph on reduced and non-reduced versions of the datasets. We set the maximum number of explored nodes to 100000 and the *noise* parameter to five percent of the number of examples in the respective dataset. For both versions of the Mutagenesis dataset (without function groups and with function groups), it took Aleph longer to produce a theory for the reduced data, however, the predictive accuracy was higher for the reduced data. In fact, the Mutagenesis dataset was the only dataset where Aleph obtained statistically significant improvement in predictive accuracy (two-sided paired t-test with $\alpha = 0.05$). For PTC dataset, reduction caused a speedup of factor 2.63. For the CAD dataset, Aleph obtained a reasonable speed-up and slightly higher predictive accuracy than for the non-reduced dataset. The reason

Table 1. Accuracy of Aleph (estimated by 10-fold cross-validation) on the reduced and non-reduced datasets. **Muta** is non-reduced Mutagenesis, **Muta-R** is reduced Mutagenesis, **Muta-FG** is Mutagenesis with function groups, **Muta-FG-R** is reduced Mutagenesis with function groups etc.

	Muta	**Muta-R**	**Muta-FG**	**Muta-FG-R**	**PTC**	**PTC-R**	**CAD**	**CAD-R**
Accu.	68.8 ± 6.6	71.6 ± 7.5	73.3 ± 11.3	73.8 ± 10.7	63.3 ± 6.7	63.6 ± 4.7	85.4 ± 9.7	88.7 ± 9.9
Run. [s]	356	378	298	339	17108	6522	244	168

why not only runtimes but also accuracies were affected by the safe reduction is that through the reduction of examples we also reduce bottom clauses, which, in turn, means smaller search space for Aleph.

Not only can we use the techniques introduced in this paper to reduce the examples, we may also use them to obtain an upper bound on the training set accuracy. Clearly, if we variabilize the constants that do not appear in the hypothesis language $\mathcal{L}$ in two examples e_1 and e_2 and obtain examples $\hat{e}_1$ and $\hat{e}_2$ and if it holds $\hat{e}_1 \preceq_E \hat{e}_2$ and $\hat{e}_2 \preceq_E \hat{e}_1$ then these two examples are indistinguishable using the hypotheses from $\mathcal{L}$. If they have different classification then we have an unremovable term added to the training set error. For example, on the PTC dataset, the calculated upper bound was 98.5%. A more complicated calculation may be used to obtain a much tighter upper bound for the particular case of hypotheses in the form of clausal theories. The bound used in this subsection applies also to propositionalization approaches.

5 Conclusions

We have shown that when a hypothesis language is fixed, even ground examples may become reducible. In other words, we have shown that, in many experiments commonly performed with ILP systems, there is a lot of redundant information in the examples. We have also shown how the underlying principles of the reduction procedure can be used to compute an upper bound on the training-set accuracy.

Acknowledgement

The authors would like to thank the anonymous reviewers of ILP'10 for their valuable comments. The authors have been supported by the Czech Grant Agency through project 103/10/1875 (*Learning from Theories*).

References

1. De Raedt, L.: Logical settings for concept-learning. Artif. Intell. 95(1), 187–201 (1997)
2. Helma, C., King, R.D., Kramer, S., Srinivasan, A.: The predictive toxicology challenge 2000-2001. Bioinformatics 17(1), 107–108 (2001)
3. Kuželka, O., Železný, F.: A restarted strategy for efficient subsumption testing. Fundam. Inf. 89(1), 95–109 (2009)

4. Maloberti, J., Suzuki, E.: An efficient algorithm for reducing clauses based on constraint satisfaction techniques. In: Camacho, R., King, R., Srinivasan, A. (eds.) ILP 2004. LNCS (LNAI), vol. 3194, pp. 234–251. Springer, Heidelberg (2004)
5. Plotkin, G.D.: A note on inductive generalization. Machine Intelligence 5, 153–163 (1970)
6. Srinivasan, A., Muggleton, S.H.: Mutagenesis: ILP experiments in a non-determinate biological domain. In: ILP 1994 (1994)
7. Žáková, M., Železný, F., Garcia-Sedano, J., Tissot, C.M., Lavrač, N., Křemen, P., Molina, J.: Relational data mining applied to virtual engineering of product designs. In: International Conference on Inductive Logic Programming (ILP 2007). Springer, Heidelberg (2007)

Appendix

Lemma 1 (Plotkin [5]). *Let A and B be clauses. If $A \preceq_\theta B$ then $A \models B$.*

Proposition 1. *Let $\mathcal{L}$ be a hypothesis language and let e be a clause. (i) Let $\tilde{e}$ be a clause obtained from e by variabilizing the constants which are not contained in the hypothesis language. Then $(H \preceq_E e) \Leftrightarrow (H \preceq_E \tilde{e})$ for any $H \in \mathcal{L}$ (ii) Let $\hat{e}$ be θ-reduction of $\tilde{e}$. Then $(H \preceq_E e) \Leftrightarrow (H \preceq_E \hat{e})$ for any $H \in \mathcal{L}$.*

Proof. We will start by showing validity of the implication $(H \models e) \Rightarrow (H \models \tilde{e})$. For contradiction, let us assume that there is a model M of the clausal theory H such that $M \models e$ and $H \not\models \tilde{e}$. Then there must be a substitution θ grounding all variables in $\tilde{e}$ such that[3] $M \models e\theta$ and $M \not\models \tilde{e}\theta$. Now, we will construct another model M' of H in which e will not be satisfied. We take each constant c in e that has been replaced by a variable V in $\tilde{e}$ and update the assignment ϕ of the constants c to objects from the domain of discourse so that $\phi(c) = \phi(V\theta)$. Clearly, we can do this for every constant c since every constant in e has been replaced by exactly one variable. Now, we clearly see that $M' \not\models e$. However, we are not done yet as it might happen that the new model with the modified ϕ would no longer be a model of H. However, this is clearly not the case since none of the constants c appears in H and therefore the change of ϕ has no effect whatsoever on whether or not H is true in M'. So, we have arrived at a contradiction. We have a model M' such that $M' \models H$ and $M' \not\models e$ which contradicts the assumption $H \models e$. The implication $(H \models e) \Leftarrow (H \models \tilde{e})$ follows directly from Lemma 1. We have $\tilde{e} \preceq_\theta e$ therefore also $\tilde{e} \models e$ and finally $H \models \tilde{e} \models e$. (ii) In order to show $(H \preceq_E e) \Rightarrow (H \preceq_E \hat{e})$, it suffices to notice that $H \preceq_E e$ and $e \preceq_\theta \hat{e}$ imply $H \preceq_E \hat{e}$. The implication $(H \preceq_E e) \Rightarrow (H \preceq_E \hat{e})$ may be shown similarly as follows: $H \preceq_E \hat{e}$ and $\hat{e} \preceq_\theta e$ imply $H \preceq_E e$.

[3] We are applying θ also to e because e need not be ground.

Learning Discriminant Rules as a Minimal Saturation Search

Matthieu Lopez, Lionel Martin, and Christel Vrain

LIFO, University of Orléans
Orléans, France
firstname.name@univ-orleans.fr

Abstract. It is well known that for certain relational learning problems, traditional top-down search falls into blind search. Recent works in Inductive Logic Programming about phase transition and crossing plateau show that no general solution can face to all these difficulties. In this context, we introduce the notion of "minimal saturation" to build non-blind refinements of hypotheses in a bidirectional approach.

We present experimental results of this approach on some benchmarks inspired by constraint satisfaction problems. These problems can be specified in first order logic but most existing ILP systems fail to learn a correct definition, especially because they fall into blind search.

1 Introduction

The main characteristics of an ILP system are given by the definition of a search space, a refinement operator and a heuristic function able to choose, at any step, the best candidate among possible refinements. In this context, the refinement operator plays a central role and approaches are usually divided into two main strategies: on one hand top-down searches start with a general clause and build specializations, on the other hand bottom-up searches start with a specific clause and generalize it. Then, in most cases, the refinement operator allows to organize the search space as a lattice, starting either from a general clause called *top* ($\top$) or a specific one called *bottom* ($\bot$). The most common top clause is defined by a clause having a literal built with the target predicate as head and an empty body; usual bottom clauses are built from a seed example and are obtained by a "saturation-like" operation [13].

In this paper, we propose a bidirectional search where the main process consists in reducing the search space. At any step, our search space is defined by a pair $(\top_i, \bot_i)$ bounding the space, and our refinement operator produces new bounds $(\top_{i+1}, \bot_{i+1})$ where $\top_{i+1}$ is more specific than $\top_i$ and $\bot_{i+1}$ is more general than $\bot_i$. The search stops when $\top_i = \bot_i$ which corresponds to the learned rule. In our approach, at any step the clause $\top_i$ can be deduced from $\bot_i$ and conversely $\bot_i$ can be deduced from $\top_i$. For this reason, our main process can be viewed either as searching for $\top_{i+1}$ from $\top_i$ (top-down) or searching for $\bot_{i+1}$ from $\bot_i$ (bottom-up).

This work has been motivated by a family of problems hard to solve for most existing ILP approaches: learning constraint problems (CP) that consists in finding a characterization of solutions (or non-solutions) of constraint problems. The goal is to generate a Constraint Satisfaction Problem (CSP) adapted to the problem to solve. We have shown in [8] that this problem can be transformed into a learning problem specified by positive and negative examples which are respectively non-solutions and solutions of the constraint problem, thus, learning the negation of the constraint problem. This point is detailed in Section 5. These examples are defined as interpretations in a subset of FOL, and additional background knowledge is provided (also described in FOL). The goal is then to learn discriminant rules, and in theory any ILP system could be used.

For example, let us consider the classical graph coloring problem expressing that a graph is well colored when two adjacent nodes have two different colors: negative (resp. positive) examples are well-colored graphs (resp. bad colored graphs), expressed in FOL, with predicates *node*, *edge*, *col*. Then a discriminative rule defining a bad-colored graph could be:

$badcol(G) \leftarrow node(G, X), node(G, Y), col(X, C), col(Y, C), edge(X, Y).$

This rule indicates that a graph is not well-colored if there exists two adjacent nodes with the same color. From a learning point of view, we can notice that this rule has a particularity. It is a discriminative one but, if we remove any literal from its body, the obtained rule has no longer a discriminative power.

The main motivation of this work is that, in practice, we observe that most systems fail on this task or find a solution after an exhaustive search. This failure can be explained since the considered problems contain specific difficulties that are well known in the ILP community, called blind search. Recent works on phase transition problems in ILP [6,7,14] and blind search or crossing "plateau" [1,3,2] indicate that searching a solution may have to cope with hard problems. These analysis show that there is no general solution to ILP learning problems, and some efforts must be done to both propose solutions and analyze their conditions of success and limitations. In this paper, we propose a new approach for learning first order clauses, and we characterize the situations where it can be successfully used.

The approach we present in this paper is based on a special saturation called "minimal saturation". The main drawback of standard saturation is that it helps defining a search space which may be very large and redundant. Our work relies on the fact that the saturation produces a clause naturally structured in layers. We propose to consider the layers one by one: at each step we search for a minimal subpart of a layer such that the associated saturation is discriminant. In doing so, the method tries to find a discriminating rule with a reduced size.

The paper is organized as follows. Section 2 introduces some basic notions whereas the formalization of the saturation used in our algorithm described in Section 3. In Section 4, we discuss the cases that seem adapted to our method and the cases that do not benefit to our search strategies. Section 5 presents the benchmarks and experiments we have made.

2 Preliminaries, Notations and Saturation

The language used to define concepts is composed of a set of terms (constants and variables) and a set of predicates. Predicates have mode declaration describing for each argument, its domain specifying what type of constants can be used: by the symbol $+$, if it is an input, or, by the symbol $-$ for an output. For example, the predicate *sum* with arity 3 and the semantic of which is X+Y=Z where X,Y and Z denote its arguments could have the mode $sum(+,+,-)$.

An atom is an expression of the form $p(t_1, t_2, \ldots, t_k)$ where p is a k-arity predicate and t_1, t_2, $\ldots$, t_k are terms. A literal is an atom or the negation of an atom. We denote by $input(l)$ and $output(l)$, the sets of terms containing respectively the inputs and outputs of a literal l. We use the same notation for a formula f: $input(f) = \bigcup_{l \in f} input(l)$ and $output(f) = \bigcup_{l \in f} output(l)$. Moreover, the set of terms composing a literal is denoted by $terms(l)$. The "variabilization" of a formula f, denoted $vars(f)$, is the operation consisting in substituting all the constants of f with variables such that each constant is replaced by a distinct variable. As said in the introduction, the saturation is a structure organized in several layers where each layer contains literals. $layer(l)$ gives the number of the layer in which the literal l appears. Finally, considering sets of positive and negative examples, $p(f)$, resp. $n(f)$, denote the number of positive and negative examples covered by a formula f.

2.1 Saturation of an Example

In this section, we present and formalize the saturation operator consisting in finding all ground literals describing an example. It has been first introduced in [13]. We present a simplified version where recall numbers[10], limiting the number of times a predicate is used are not considered. We propose to illustrate the construction with the following toy example, close to Michalski's trains:

Example

In this example, each train is composed of cars. Each car has different features as its number of wheels and its size. A car carries geometrical objects, described by their shape and their quantities. For instance, two examples corresponding to a positive and a negative example of a target concept are described as follows:

$$\begin{aligned} t_1^+ : \{&has\_car(t_1^+, c_1), has\_car(t_1^+, c_2), has\_car(t_1^+, c_3), wheels(c_1, 2),\\ &wheels(c_2, 3), wheels(c_3, 5), long(c_1), long(c_2), long(c_3),\\ &load(c_1, circle, 3), load(c_2, circle, 6), load(c_3, triangle, 10)\} \end{aligned}$$

$$\begin{aligned} t_2^- : \{&has\_car(t_2^-, c_1), has\_car(t_2^-, c_2), has\_car(t_2^-, c_3), wheels(c_1, 1),\\ &wheels(c_2, 4), wheels(c_3, 3), long(c_1), long(c_2), short(c_3),\\ &load(c_1, circle, 4), load(c_1, rectangle, 2), load(c_2, triangle, 5)\\ &load(c_2, circle, 3), load(c_3, circle, 2)\} \end{aligned}$$

Let us suppose that the modes are defined by: $has\_car(+,-)$, $long(+)$, $short(+)$, $wheels(+,-)$ *and* $load(+,-,-)$. *Let us assume that the target definition corresponds to trains having at least two cars containing objects with the same shape* *and such that one of the two has a greater number of objects and a greater number of wheels than the other:*

An example is represented with the predicate goodtrain with a single argument which is the ID of train, as for instances : $goodtrain(t_1^+)$. Literals describing the train are then given as ground facts in the background knowledge as it is the case in several systems like Aleph. The background knowledge also contains intensional predicates described by of Horn clauses.

$$\begin{aligned} goodtrain(T) :-& has\_car(T, C_1), has\_car(T, C_2), C_1 \neq C_2, \\ & wheels(C_1, W_1), wheels(C_2, W_2), \\ & load(C_1, O, L_1), load(C_2, O, L_2), \\ & W_1 < W_2, L_1 < L_2 \end{aligned}$$

The saturation of an example is based on information given by domains and modes of predicates. It is obtained by adding as many literals as possible *w.r.t.* the background knowledge such that all literals form a connected formula. The set of literals, composing the saturation, is organized in ordered layers: a literal is in a layer k if all the input terms needed to its introduction have already been introduced in previous layers, and at least one of them has been introduced in the layer $k-1$. The saturation could be large and in the state-of-the-art, it is often parameterized by a maximal depth: the building steps when this depth is reached. This maximal depth is denoted by i.

Before defining the saturation, we introduce following notation: $litsOfLayer(S, k)$ denotes the set of literals of a sets belonging to a layer k. It is defined by: $litsOfLayer(k) = \{l \mid layer(l) = k\}$

To define the saturation, we first define a layer k, called $sat_k(S_l)$, where S_l is a set of literals describing previous layers (all layers $j < k$):

$$\begin{aligned} sat_k(S_l) = \{l \mid & input(l) \subseteq terms(S_l) \\ & \wedge l \notin S_l \\ & \wedge input(l) \cap output(litsOfLayer(S_l, k-1)) \neq \emptyset\} \end{aligned}$$

Then, we can formalize, given a set S_l of literals introduced in layers before k, the set of literals belonging to layers k, $k+1$, ..., i:

$$\begin{array}{ll} sat(S_l, k, i) = sat_k(S_l) \cup sat(S_l, k+1, i) & 1 \leq k \leq i \\ sat(S_l, k, i) = \emptyset & k > i \end{array}$$

Finally, the saturation of an example e of a target concept p with the maximal layer i can be written:

$$sat(p(e), i) = \{p(e)\} \cup sat(\{p(e)\}, 1, i)$$

Let us remark that the predicate p has the mode: $p(-)$.

Example (cont.)

We consider that we have in the background knowledge, along the description of the examples, the predicate $\neq$ comparing cars and geometrical forms with mode $\neq (+,+)$, and the predicate $<$ comparing either numbers of wheels or numbers of object with the mode $< (+,+)$.

The saturation of the example t_1^+ with the maximal depth equal to 3 is:

Layer	*Literals*
0	$goodtrain(t_1^+)$
1	$has\_car(t_1^+, c_1), has\_car(t_1^+, c_2), has\_car(t_1^+, c_3)$
2	$wheels(c_1, 2), wheels(c_2, 3), wheels(c_3, 4),$ $long(c_1), long(c_2), long(c_3),$ $load(c_1, circle, 3), load(c_2, circle, 5), load(c_3, triangle, 10),$ $c_1 \neq c_2, c_1 \neq c_3, c_2 \neq c_3$
3	$2 < 3, 2 < 4, 3 < 4,$ $triangle \neq circle,$ $3 < 5, 3 < 10, 5 < 10$

3 Minimal Saturation to Search

In this section, we present our rule learning algorithm. A search state is a set of literals corresponding to a clause. The states are organized as a lattice with two bounds $\top$ and $\bot$, and a partial order to compare the generality of search states. For the partial order, we say that a state s_1 is more general than an other s_2 if s_1 is a subset of s_2 (with possible renaming of variables). The search is biased using a positive example s, called a seed. Considering the target concept p, we set $\top$ to $\{p(s)\}$ and $\bot$ to $sat(p(s), i)$ where i is the maximal layer of the saturation. This search space is often used in the state-of-the-art algorithms.

With the assumption that $\bot = sat(\top, i)$ rejects all negative examples, the search space contains at least one rule discriminating negative examples. It covers at least one positive example, the seed. The goal is to discover a better rule than $\bot$, *i.e.* a rule that covers more positive examples and always no negative ones. Given a step k in the learning algorithm, we aim to set the layer k such that this layer would be shorter as possible. When the assumption is not satisfied, the refinement algorithm throws an exception to specify the insolubility of the learning problem.

Our learning algorithm is a bidirectional search where we progressively refine two hypothesis $H_\top$ and $H_\bot$ keeping the relation $H_\bot = sat(H_\top, i)$ always correct.

This section is divided into two parts: first we present our refinement operator and then, the algorithm, iterating the refinement steps.

3.1 Refinement Operator

Let us first recall that we have two hypothesis $H_\top$ and $H_\bot$ with the relation $H_\bot = sat(H_\top, i)$ where i denotes the maximal depth of the saturation. For the kth application of the refinement operator, it aims at building a new couple of hypothesis $(H'_\top, H'_\bot)$ such that $H'_\bot = sat(H'_\bot, i)$ and such that the layer k is a minimal subset of literals from the layer k of $H_\bot$.

This operator searches for a subset of literals of the layer k, such that the corresponding $H_\bot$ discriminates some positive examples from all the negative examples, this subset will be added to $H_\top$.

To formalize the refinement operator, we start by defining the set of candidate couples $(H'_\top, H'_\bot)$ that are correct refinements of a current pair of hypotheses $(H_\top, H_\bot)$. Let $\rho(H_\top, H_\bot, k, i)$ denote this set, where k represents the layer wherein the refinement searches for a subset and i the maximal depth:

$$\begin{aligned}\rho(H_\top, H_\bot, k, i) = \{(H'_\top, H'_\bot) \mid\ & H'_\top = H_\top \cup S_k \\ & \wedge S_k \subset litsOfLayer(H_\bot, k) \\ & \wedge H'_\bot = sat(H'_\top, k+1, i) \\ & \wedge n(H'_\bot) = 0\}\end{aligned}$$

Let us notice that when $S_k = \emptyset$, $H'_\top = H'_\bot$ ($H'\top$ have no literals in layer k and therefore subsequent layers are empty).

This set contains only couples $(H'_\top, H'_\bot)$ where $H'_\top$ is more specific than $H_\top$, $H'_\bot$ is more general than $H_\bot$ and $H'_\bot$ rejects all negative examples.

To determine the subset of literals, we search the smallest one with a greedy breadth first search as explained in Section 3.2.

3.2 Refinements Selection

In this section, we described how we choose among all the refinements in ρ. Our strategy is to select the smallest one in term of S_k, *i.e.* a hypothesis $H_\top$ with the smallest number of literals of layer k. If there are several ones, we return the one covering the maximum number of positive examples. The number of candidates is exponential comparing to the size of $litsOfLayer(H_\bot, k)$. To avoid to generate all of them, we search the smallest one with a breadth-first search. The refinement algorithm can be written as follows:

Algorithm : $refine(H_\top, H_\bot, k, i)$

```
1.  for $j$ from 0 to $|litsOfLayer(H_\bot, k)|$
2.     $S\rho = \emptyset$
3.     for each subset $S_k$ of $litsOfLayer(H_\bot, k)$ s.t. $|S_k| = j$
4.        $H'_\top = H_\top \cup S_k$
5.        $H'_\bot = H'_\top \cup sat(H'_\top, k+1, i)$
6.        if $n(H'_\bot) = 0$ then
7.           $S\rho = S\rho \cup \{(H'_\top, H'_\bot)\}$
8.    if $S\rho \neq \emptyset$ then
9.       return $argmax_{(H'_\top, H'_\bot) \in S\rho} p(H'_\bot)$
10. throw inconsistent bottom clause
```

We can notice that the algorithm starts with the empty subset (for $j = 0$). In this case, $H_\top = H'_\top = H'_\bot$, and if this rule is correct then the rule learning algorithm will stop since there are no literals in layers up to k (See section 3.3).

Example (cont.)

To illustrate the refinement operator, let us consider as the current couple of hypotheses : $(H_\top, H_\bot) = (vars(\{goodtrain(t_1^+)\}), vars(sat(\{goodtrain(t_1^+)\}, 3))$ *and then the refinement produced with* $refine(H_\top, H_\bot, 1, 3)$. $H_\bot$ *corresponds to the* "variabilized" *version of the saturation computed in the previous example in section 2.1.*

The algorithm begins with subset of the first layer of size 1. All subsets produce the same couple with a correct substitution and we obtain for $\rho(H_\top, H_\bot, 1, 3)$ *such that* $|S_k| = 1$*, the unique couple:*

Layer	*Literals*
0	$\mathbf{goodtrain}(\mathbf{V_1})$
1	$\mathbf{has\_car}(\mathbf{V_1}, \mathbf{V_2})$
2	$wheels(V_2, V_3)$, $load(V_2, V_4, V_5)$, $long(V_2)$
3	

where $H'_\top$ *is in bold characters and* $H'_\bot$ *contains the saturation of* $H'_\top$ *(layers 0 to 3 in the array). This couple is not correct because* $H'_\bot$ *covers the negative example* t_2^-*.*

*With subsets of size 2 (*i.e. $|S_k| = 2$*),* $\rho(H_\top, H_\bot, 1, 3)$ *produces:*

Layer	*Literals*
0	$\mathbf{goodtrain}(\mathbf{V_1})$
1	$\mathbf{has\_car}(\mathbf{V_1}, \mathbf{V_2}), \mathbf{has\_car}(\mathbf{V_1}, \mathbf{V_6})$
2	$wheels(V_2, V_3), wheels(V_6, V_7)$, $load(V_2, V_4, V_5), load(V_6, V_4, V_8)$, $long(V_2), long(V_6)$, $V_2 \neq V_6$
3	$V_3 < V_7, V_5 < V_8$

Layer	*Literals*
0	$\mathbf{goodtrain}(\mathbf{V_1})$
1	$\mathbf{has\_car}(\mathbf{V_1}, \mathbf{V_2}), \mathbf{has\_car}(\mathbf{V_1}, \mathbf{V_9})$
2	$wheels(V_2, V_3), wheels(V_9, V_10)$, $long(V_2), long(V_9)$, $load(V_2, V_4, V_5), load(V_9, V_11, V_12)$, $V_2 \neq V_9$
3	$V_3 < V_10, V_11 \neq V_4, V_5 < V_12$

Just the first one rejects the negative example and the corresponding couple of hypotheses will be returned by the refining algorithm.

3.3 Learning a Rule

An interesting rule is a rule rejecting all negative examples (consistency) and covering a maximum number of positive examples. With our approach, the rule we compute could not be the better one in terms of the number of covered positive examples, because we search for minimal layers to limit the complexity of our refinement algorithm. During the bidirectional search, we intend to progressively refine the couple of hypotheses $(H_\top, H_\bot)$, until the two hypotheses converge ($H_\top = H_\bot$). In order to obtain it, we progressively refine each layer starting from $k = 1$ to $k = i$ where i is the maximal layer. We initialize the search with a couple corresponding to a seed example s and its saturation, where constants are replaced by variables. The following algorithm describes the search for a rule for the target concept p, where the search space is biased by the seed s:

```
Algorithm : learnrule(p, s, i)
1.  $H_\top = \{vars(p(s))\}$
2.  $H_\bot = vars(sat(p(s), i))$
3.  for each layer $k$ from 1 to $i$
4.      $(H_\top, H_\bot) = refine(H_\top, H_\bot, k, i)$
5.  return $H_\top$
```

4 Advantages and Limitations

Our algorithm has been motivated by specific applications of Machine Learning to Constraints Modelling. To compute a refinement, it searches among the subsets of literals contained in a layer the smallest one that corresponds to a bottom hypothesis rejecting all negative examples. To be efficient our algorithm must stop before enumerating all the subsets. This means that it is dedicated to problems characterized by the following properties: the saturations of the examples can be structured into several layers and there exists discriminating rules involving few literals in each layer. For instance, it is not adapted to the classical mutagenesis benchmark [9]: the saturation of an example is composed of only two layers, most of the literals of the discriminating rules occurring in the first layer.

The learning problems we are interested in are difficult for top-down approaches because search is blind: a choice must be made between refinements with the same heuristic values. In some cases, this can be avoided by considering bottom-up approaches. However, the complexity of the coverage test for examples depends on the size of the saturation, becoming too expensive when the saturation is large. Our method intends to avoid the "*plateau*" phenomenon for top-down approaches evaluating a rule in a bottom-up way (by means of its saturation) but limiting the size of hypotheses in a top-down way. Preferring a bidirectional search, our aim is to build a minimal saturation, which can be easily tested on examples while guiding the search.

5 Application to Constraint Satisfaction Problem and Experiments

Constraint Programming (CP) is a very successful formalism to model and to solve a wide range of decision problems, from arithmetic puzzles to timetabling and industrial scheduling problems. One of our motivations for this work is to learn CSP models described in a first order language, defining the solutions of a

Graph coloring problem

$$n(X), n(Y), col(X, A) \wedge col(Y, B) \rightarrow A \neq B \ \vee \ \neg adj(X, Y)$$

Simplified school timetable

$$\begin{gathered} timetable(L_1, T_1, R_1, S_1) \wedge timetable(L_2, T_2, R_2, S_2) \\ \rightarrow L_1 = L_2 \ \vee \ R_1 \neq R_2 \ \vee \ S_1 \neq S_2 \\ \bigwedge \\ timetable(L_1, T_1, R_1, S_1) \wedge timetable(L_2, T_2, R_2, S_2) \\ \rightarrow T_1 \neq T_2 \ \vee \ L_1 = L_2 \ \vee \ S_1 \neq S_2 \end{gathered}$$

Simplified jobshop

$$\begin{gathered} schedule(J, T, B, E, M) \rightarrow B < E \\ \bigwedge \\ schedule(J_1, T_1, B_1, E_1, M_1) \wedge schedule(J_2, T_2, B_2, E_2, M_2) \\ \rightarrow J_1 = J_2 \ \vee \ M_1 \neq M_2 \ \vee \ B_1 > E_2 \ \vee \ E_1 < B_2 \\ \bigwedge \\ schedule(J_1, T_1, B_1, E_1, M_1) \wedge schedule(J_2, T_2, B_2, E_2, M_2) \\ \rightarrow J_1 = J_2 \ \vee \ E_1 < B_2 \ \vee \ prev(T_1, T_2) \end{gathered}$$

N-queens problem

$$\begin{gathered} position(Q_1, L_1, C_1) \wedge position(Q_2, L_2, C_2) \rightarrow Q_1 = Q_2 \vee L_1 \neq L_2 \\ \bigwedge \\ position(Q_1, L_1, C_1) \wedge position(Q_2, L_2, C_2) \rightarrow Q_1 = Q_2 \vee C_1 \neq C_2 \\ \bigwedge \\ position(Q_1, L_1, C_1) \wedge position(Q_2, L_2, C_2) \wedge gap(L_1, L_2, I_1) \wedge gap(C_1, C_2, I_2) \\ \rightarrow Q_1 = Q_2 \vee I_1 \neq I_2 \end{gathered}$$

Fig. 1. Some examples: all variables are universally quantified

CSP[8]. We have shown that this can be transformed into the problem of learning a logic program of the negative concept, composed of the non-solutions of the CSP. The process is detailed in[8], but let us illustrate it by our graph coloring problem. Finding a color for each vertex such that two adjacent vertices do not have the same color can be described by the rule:

$$node(X) \wedge node(Y) \wedge col(X, A) \wedge col(Y, B) \rightarrow A \neq B \ \vee \ \neg adj(X, Y)$$

Considering the negative concept of bad-colored graphs, adding a target concept $badcol(G)$, where G is the ID of a graph and adding G also to the extensional predicates, we obtain the following datalog rule:

$$\begin{aligned} badcol(G) \leftarrow\ & node(G, X) \wedge node(G, Y) \\ & \wedge col(G, X, A) \wedge col(G, Y, B) \\ & \wedge A = B \wedge adj(F, X, Y) \end{aligned}$$

	Propal			Our algorithm		
benchmark	# learned rules	time (s)	acc.	# learned rules	time (s)	acc.
Graph coloring	1	0	100%	1	0.17	100%
School timetable	3	11	98,33%	2	0.69	100%
Job-shop	6	103	87,78%	5	7.37	100%
N-queens	-	-	-	3	29.11	100%
	Aleph1			Aleph2		
Graph coloring	1	0.24	100%	1	0.14	100%
School timetable	1	1.24	100%	1	0.31	100%
Job-shop	3	1051.03	100%	6	1130.88	96%
N-queens	3	489.49	100%	3	4583.84	61.67%

Fig. 2. Experiments with CSP benchmarks

This explains why we focus on the negative concept of the constraint problem. Figure 1 presents classical CSP problems and the models we aim at learning. Let us notice that these models are independent of the size of the CSP (number of queens for instance). We have built random benchmarks for these problems. In doing so, we have generated a set of solutions and non-solutions. To produce solutions, we have chosen a random size for each problem (*e.g.* for the graph coloring problem, the number of vertices and colors), and then solved the CSP. For negative ones, we have proceeded in a similar way but constraints have been relaxed and we have checked that there is at least one unsatisfied constraint. The examples have been produced using a random heuristic when solving the CSP.

We have tested different state-of-the-art ILP systems on these benchmarks: Foil[11], Beth[16], Aleph[15], ICL[12], Propal[5]. Foil performs a *Top-down* search starting from the most general hypothesis and progressively specializing by adding new literals *w.r.t.* the θ-subsumption. As in our work, Beth keeps a kind of bottom clause in addition to a hypothesis, but it is used to avoid a complete computation of the saturation whereas ours attends to bound the search space and is used to evaluate refinements. Aleph is a large ILP system allowing the use of various top-down or bottom-up strategies. ICL searches for a theory in a disjunctive normal form with a beam search. Propal uses a data driven strategy, *i.e.* the refinement operator uses a negative example that must be rejected in the possible refinements. Only Propal (the version described in [4], faster than the version given in [5]) and Aleph with certain configurations have succeeded. The first configuration of Aleph, called Aleph1, corresponds to a breadth-first search with a maximum of 200 000 visited search nodes and an infinite open list. The second one, called Aleph2, only differs from the search strategy which is a heuristic search. We use the default heuristic of Aleph, called *coverage*, and consisting in the difference between positive examples and negative ones[1]. We have implemented a prototype for our learning algorithm and Figure 2 presents the results obtained with these systems. The more complicated

[1] We have tested all the heuristics implemented in Aleph with similar results

the target concept, the more Propal and Aleph2 find incorrect theory. For the n-queens problems, Propal has been stopped after ten hours. This illustrates the difficulty of top-down approaches when search is blind. Moreover, Propal has to face combinatorial explosion when searching for near-misses. Aleph1 succeeds with all the benchmarks. However, significant computation times is required for complicated benchmarks. Our method has succeeded on all the benchmarks: it finds accurate theories in a short amount of times. These good results can be explained by the structure of the target concepts as described in Section 4.

Our prototype, as well as the benchmarks, can be obtained by sending a mail to the authors.

6 Conclusion

In this paper, we have presented a new learning algorithm to build logic programs. Our method aims at finding a minimal saturation. It consisting in finding, for each layer of the saturation of a seed, a minimal subset of literals such that the obtained rule discriminates some positive examples from negative ones. This approach is not always efficient, but in certain cases where other state-of-the-art algorithms fail, it can succeed avoiding the blindness faced by top-down strategies and the consuming coverage tests with bottom-up strategies. Our approach is efficient when the saturation of seed examples produces a bottom clause structured in several layers and when there exists a discriminating rule involving few literals of each layer. We experiment our algorithm on learning problems inspired by constraint programming. These problems present aforesaid features and our algorithm succeeds when other systems fail. These results are encouraging and we aim at testing on other benchmarks.

In future works, we wish to improve the refinement operator by replacing the greedy breadth-first search for layer subsets. This part of our algorithm is the principal limitation of our approach. For example, we could guide the search for subsets with classical heuristics or using a beam search.

Another improvement would be to handle numeric variables. So, we could test our learning algorithm with a large range of acknowledged learning problems.

References

1. Alphonse, É., Osmani, A.: A model to study phase transition and plateaus in relational learning. In: Železný, F., Lavrač, N. (eds.) ILP 2008. LNCS (LNAI), vol. 5194, pp. 6–23. Springer, Heidelberg (2008)
2. Alphonse, É., Osmani, A.: On the connection between the phase transition of the covering test and the learning success rate in ilp. Machine Learning 70(2-3), 135–150 (2008)
3. Alphonse, E., Osmani, A.: Empirical study of relational learning algorithms in the phase transition framework. In: Buntine, W., Grobelnik, M., Mladenić, D., Shawe-Taylor, J. (eds.) ECML PKDD 2009. LNCS, vol. 5781, pp. 51–66. Springer, Heidelberg (2009)

4. Alphonse, É., Rouveirol, C.: Lazy propositionalisation for relational learning. In: Horn, W. (ed.) ECAI, pp. 256–260. IOS Press, Amsterdam (2000)
5. Alphonse, É., Rouveirol, C.: Extension of the top-down data-driven strategy to ILP. In: Muggleton, S.H., Otero, R.P., Tamaddoni-Nezhad, A. (eds.) ILP 2006. LNCS (LNAI), vol. 4455, pp. 49–63. Springer, Heidelberg (2007)
6. Botta, M., Giordana, A., Saitta, L., Sebag, M.: Relational learning: Hard problems and phase transitions. In: Lamma, E., Mello, P. (eds.) AI*IA 1999. LNCS (LNAI), vol. 1792, pp. 178–189. Springer, Heidelberg (2000)
7. 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)
8. Lallouet, A., Lopez, M., Martin, L., Vrain, C.: On learning constraint problems. In: 22th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2010 (2010)
9. Lodhi, H., Muggleton, S.: Is mutagenesis still challenging. In: ILP - Late-Breaking Papers (2005)
10. Muggleton, S.: Inverse entailment and progol. New Generation Computing, Special issue on Inductive Logic Programming 13(3-4), 245–286 (1995)
11. Ross Quinlan, J., Mike Cameron-Jones, R.: Foil: A midterm report. In: Brazdil, P.B. (ed.) ECML 1993. LNCS, vol. 667, Springer, Heidelberg (1993)
12. De Raedt, L., Van Laer, W.: Inductive constraint logic. In: ALT, pp. 80–94 (1995)
13. Rouveirol, C.: Extensions of inversion of resolution applied to theory completion. In: Muggleton, S. (ed.) ILP, pp. 63–92. AP (1992)
14. Serra, A., Giordana, A., Saitta, L.: Learning on the phase transition edge. In: IJCAI, pp. 921–926 (2001)
15. Srinivasan, A.: A learning engine for proposing hypotheses (Aleph), `http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/aleph.html`
16. Tang, L.P.R.: Integrating top-down and bottom-up approaches in inductive logic programming: applications in natural language processing and relational data mining. PhD thesis, University of Texas, Supervisor-Mooney (2003)

Variation of Background Knowledge in an Industrial Application of ILP

Stephen H. Muggleton[1], Jianzhong Chen[1], Hiroaki Watanabe[1], Stuart J. Dunbar[2], Charles Baxter[2], Richard Currie[2], José Domingo Salazar[2], Jan Taubert[3], and Michael J.E. Sternberg[3]

[1] Imperial College London
[2] Syngenta Ltd
[3] BBSRC Rothamsted Research

Abstract. In several recent papers ILP has been applied to Systems Biology problems, in which it has been used to fill gaps in the descriptions of biological networks. In the present paper we describe two new applications of this type in the area of plant biology. These applications are of particular interest to the agrochemical industry in which improvements in plant strains can have benefits for modelling crop development. The background knowledge in these applications is extensive and is derived from public databases in a Prolog format using a new system called Ondex (developers BBSRC Rothamsted). In this paper we explore the question of how much of this background knowledge it is beneficial to include, taking into account accuracy increases versus increases in learning time. The results indicate that relatively shallow background knowledge is needed to achieve maximum accuracy.

1 Introduction

Systems Biology is a rapidly evolving discipline that seeks to determine how complex biological systems function [6]. It works by integrating experimentally derived information with mathematical and computational models. Through an iterative process of experimentation and modelling, Systems Biology aims to understand how individual components interact to govern the functioning of the system as a whole. In several recent papers [13,3], Inductive Logic Programing (ILP) has been applied to Systems Biology problems, in which it has been used to fill gaps in the descriptions of biological networks.

Two new industrial applications of this type in the area of plant biology are being explored within the Syngenta University Innovation Centre (see Section 2). This centre of excellence in Systems Biology aims to address biological research questions related to the improvement of plant strains involved in crops. The centre uses mathematical and computational modelling techniques developed at Imperial College London [2]. The background knowledge in these applications is generated by a system called Ondex [7]. Ondex is unique in its ability to generate large amounts of Prolog background knowledge on cell biochemistry by parsing, filtering and combining various publicly-available databases.

In this paper we use Ondex to explore the performance effects of varying the amount of background knowledge available to an ILP learning engine. This is done by generating variations of background knowledge using the Relation Neighbours Filter in Ondex together with a tehnique for sampling Relations. The effects of varying the background knowledge are measured on both learning time and predictive accuracy. The experimental results indicate that while learning time increases monotonically with the amount of background knowledge, relatively shallow degree of background knowledge is required to achieve maximum accuracy.

The paper is arranged as follows. In Section 2 we introduce the applications on which the experiments were conducted. We then describe the Ondex system for generating the background knowledge in Section 3. The experiments are then described in Section 4. Finally we conclude and describe further work in Section 5.

2 Application Descriptions

2.1 University Innovation Centre (UIC) Overview

Agricultural research relies on understanding interactions of genes and chemicals in a biological context. The search for a new biological trait to use in a conventional or genetic modification breeding programme is complex. It can take up to ten years and millions of dollars to bring such a development to market. The same is true for the search for new agrochemicals. Part of this development process is an assessment of the safety of the gene or chemical to the environment and its potential toxicity to both mammals and beneficial organisms. Systems Biology takes a new, integrated, approach to address these important challenges. Syngenta [1] is a leading Agrichemical company with a number one position in chemicals and is number three in high value seeds. Syngenta has established a "University Innovation Centre" (UIC) on Systems Biology at Imperial College London [2] to implement a "systems approach" to agricultural research. The centre has begun with two pioneer projects, tomato ripening and predictive toxicology.

2.2 Tomato Application

The characteristics of the tomato fruit that reaches the consumer are defined by the combination of its biochemical and textural properties. Metabolic components (volatiles, pigments, sugars and amino acids) define the appearance and flavour whilst structural properties (cell adhesion, cell size, cuticle thickness, water content) define mouth-feel and texture perception. Together these components determine fruit quality and are crucial in influencing the success of commercial varieties.

At the genetic and biochemical level the regulation of fruit development and ripening remains poorly understood. In this project we are applying ILP to deepen our understanding of the metabolic processes controlling tomato fruit development. By applying machine learning techniques to transcript and metabolite profiling data we are developing metabolic networks and building a predictive model of tomato ripening and fruit quality. Through the coordinated analysis of gene expression and metabolite changes across fruit development we aim to identify new genetic targets that play a role in controlling the ripening process. Such knowledge will allow us to focus on these genetic control points in breeding new tomato varieties, thus producing the most favourable combination of fruit quality characters in the ripe fruit.

2.3 Predictive Toxicology

An assessment of the potential to cause cancer is a key component of the risk assessment on a new Crop Protection Active Ingredient. The two year and 80 week bioassays in rats and mice, respectively, provide the Hazard Information to evaluate this risk. If tumours are observed in these trials, an assessment of their relevance for human risk may then be required. This typically makes use of the IPCS/HESI Human Relevance Framework, where the first step is the development of a mode-of-action case to describe the series of causal key events that lead to rodent tumours. The second step is to examine the plausibility of these key events occurring in humans and so guide an assessment of the relevance of the rodent findings.

This project aims to build a model that integrates the metabolic and gene expression regulatory networks that underlie initial key events in liver tumour promotion induced by model non-genotoxic carcinogens. It is envisaged that cycles of hypothesis generation informed by model building and experimental testing will allow the identification of those regulatory components that are key components in liver tumour promoting modes of action. Ultimately this will allow us to improve mechanistic understanding and so provide key data to explain the basis of the thresholds in dose and species specificity in response, thereby allowing more informed human health risk assessments.

3 Ondex: A Biological Background Knowledge Generator

Data integration in the life sciences still remains a significant challenge for bioinformatics [5]. Rather than developing a bespoke data integration solution for assembling the background knowledge for the machine learning task, the open source data integration framework Ondex [7] was selected as a general solution to bringing all the required data together. Ondex uses a graph-based approach with a data warehouse for integrating biological data. The nodes in the graph represent biological concepts, e.g. enzymes and metabolites. Edges in the graph represent relations between biological concepts, e.g. a set of enzymes catalyses a biochemical reaction. Both nodes and edges in the graph can have additional attributes, e.g. an enzyme name or an amino acid sequence. One of the reasons for choosing the Ondex system as a background knowledge generator is the natural correspondence between its graph representation and the requirement of generating background knowledge as Prolog clauses for ILP learning.

Data from key biological pathway and gene function information resources including KEGG [11], LycoCyc [4] were transformed into a semantically consistent graph representation using Ondex. In order to create a non-redundant and coherent knowledge base, mapping methods were used to identify equivalent and similar entities among the different data sources. Once the databases were integrated, the resulting knowledge base was available for further analysis and visualization using the graph-based methods built in Ondex.

A key feature of the Ondex user client is that it allows the extraction of sub-graphs based on certain criteria. For example, it is simple to extract sub-graphs selected on by the class of biological concepts or relations, or where concepts posses a particular attribute, or from a graph-neighbourhood around particular nodes of interest. Such criteria can be combined in a workflow to manipulate the information to be included into the background knowledge. In order to support ILP learning, a general background knowledge generating utility was built that respected an agreed Prolog syntax, with Ondex concept class names and relation type names becoming predicate symbols and attributes becoming Prolog term structures. Every concept and relation was given a unique ID as the first argument in each predicate, which was used by other predicates to associate attributes with concepts and relations. The translation of attributes of concepts and relations were defined using the Ondex-generated Prolog code export utility. By following agreed conventions it was also possible to translate Prolog format back into an Ondex graph, thus enabling the results of the machine learning process to be imported back into Ondex where they could be visualised in the context of the original knowledge base.

4 Experiments

Two independent experiments were conducted in the study to empirically investigate the *null hypothesis*: variations of background knowledge (BK) do not lead to increased predictive accuracy.

4.1 Materials and Methods

Tomato Application. An initial ground background knowledge base was derived from *LycoCyc* database [4,9] and exported as Prolog format using the Ondex system. The knowledge base depicts the relational structure of tomato biological network (shown in Fig. 1), including the fundamental components, e.g. *compounds, reactions, enzymes*, and their relations, e.g. *consumed_by, produced_by, catalysed_by*, etc. Two types of raw data were provided by the domain experts in the experiments using gene mutants to study altered tomato ripening - concentration changes of metabolites and gene transcripts for four genotypes during 13 time slices. The data were expressed in terms of binary

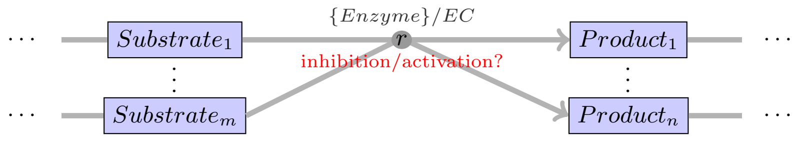


Fig. 1. Illustration of the relational background knowledge, where a set of compounds $\{Substrate\}_1^m$ are consumed by a reaction r, which produce a set of compounds $\{Product\}_1^n$; r is catalysed by a set of enzymes with an EC number $\{Enzyme\}/EC$ which is aggregated from a set of gene transcripts; the ILP learning is to abduce inhibition/activation occured in r.

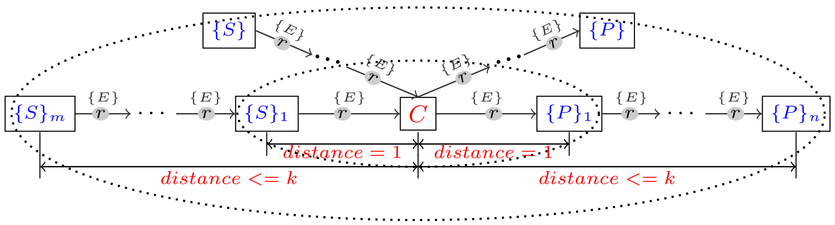


Fig. 2. Illustration of biological network structure and k-compound-neighbours (k-cn, $k \geq 1$) of a centroid compound C, where $\{S\}, \{P\}, \{E\}, r$ stand for a set of substrate compounds, product compounds, enzymes and a reaction, respectively

(up/down-regulation) for the purposes of applying ILP. In order to deal with the many-to-one relationships among gene transcripts, enzymes and reactions, a set of relevant transcriptomic data were further 'compressed' into one value using *SUM aggregate* for a reaction. Three datasets were chosen for modeling *tomato aspartate and the connected subnetwork*. Each contains the concentration changes of 10 metabolites as learning examples and 16 transcripts as observed facts on a particular time point[1].

Variations of the background knowledge were generated using the Ondex *relation neighbours filter* (RNF), which extracts a subset from an original network given a set of centroid nodes and some distance k. In our experiments, the subnetwork consists of k-compound-neighbours (k-cn, as illustrated in Fig. 2) , which is defined based on a corresponding k-reaction-neighbours (k-rn) as follows.

$$0\text{-cn} = \{\text{centroid compounds}\}$$

$$k\text{-rn} = \bigcup_{c \in (k-1)\text{-cn}} (r \mid \text{consumed\_by}(c, r)) + \bigcup_{c \in (k-1)\text{-cn}} (r \mid \text{produced\_by}(c, r))$$

$$k\text{-cn} = \bigcup_{r \in k\text{-rn}} (c \mid \text{consumed\_by}(c, r)) + \bigcup_{r \in k\text{-rn}} (c \mid \text{produced\_by}(c, r))$$

[1] They respectively represent three time points - 15 days post anthesis, the breaker point when the fruit starts to change colour, and 7 days after the breaker stage.

Table 1. Algorithm of sampling relation neighbours filter (SRNF)

```
1.0-cn={learning examples};
2.for each distance $k = 1, \ldots, K$, where $K$ is a distance threshold corresponding to
    the original background knowledge base
  2.1.$k$-rn={}, for each $c \in (k-1)$-cn
    $k$-rn $= k$-rn $\cup \{r \mid \text{consumed\_by}(c,r) \vee \text{produced\_by}(c,r)\}$;
  2.2.$ks$-rn = sample($k$-rn,$sr$), where $sr$ is a manually set sampling rate;
  2.3.$k$-cn={}, for each $r \in ks$-rn
    $k$-cn $= k$-cn $\cup \{c \mid \text{consumed\_by}(c,r) \vee \text{produced\_by}(c,r)\}$;
  2.4.output $k$-cn.
```

Furthermore, a *sampling relation neighbours filter* (SRNF) is developed in order to generate a series of evenly varied variations, containing sampled subsets of k-rn in which the size of k-cn can be controlled by a given sampling rate. The algorithm of SRNF is shown in Table 1.

The datasets and variations of ground background knowledge were given to the abductive ILP [13] system Progol5.0 [10] together with a set of non-ground rules, which describe the underlying transitive behaviour of concentrations of metabolites and enzymes. Progol5.0 was then required to derive *inhibition* on reactions. Predictive performance was tested and evaluated against variations of the background knowledge (k-cn).

Predictive Toxicology Application. In the Predictive Toxicology application, the metabolomic and genomic data reported in [14] were integrated with the KEGG Rattus norvegicus (rat) database using Ondex. The data integration was performed in the following four steps. The first step was to extract relational information from the KEGG database using the Ondex KEGG parser. After parsing an XML version of KEGG database, unnecessary information was discarded using Ondex's filtering functions. We constructed an initial model by keeping the 6 meta concepts (Gene, Protein, Enzyme, Enzyme Classification, Reaction and Compound) and 6 relations as shown in Figure 3. Note that a meta concept is a set of unique concepts. For example, the meta concept *Compound* is a set of compounds collected in KEGG COMPOUND database.

The second step was to define new relations at the meta-concept level in order to build our own models. In this paper we design mappings from *Gene* to *Enzyme Classification* (EC) to map the effects of the gene expressions at the EC level. The following shows a chain of meta concepts in Prolog.

$$\begin{aligned} project(Gene, EC) \quad :- \; & is\_encoded\_by(Protein, Gene), is\_a(Enzyme, Protein), \\ & catalyzed\_by(Reaction, Enzyme), \\ & part\_of\_catalyzing\_class(Reaction, EC) \end{aligned}$$

In the above formula, the new predicate, *project/2*, can be used to define the projection from *Gene* to *EC*. In the Predictive Toxicology application, this projection forms a many-to-one mapping. Part of the projections is included in Appendix 1.

Relations	From	To
is encoded by	Protein	Gene
is a	Enzyme	Protein
catalyzed by	Reaction	Enzyme
part of catalyzing class	Reaction	Enzyme Classification
consumed by	Compound	Reaction
produced by	Compound	Reaction

Fig. 3. The 6 meta classes and with their associated relations

The third step was to integrate the parsed Ondex model with the numerical gene expression data at the EC level. We applied the Ondex *tab delimited file* parser and stored the parsed numerical data as attribute values in the associated concepts. The gene expression data at day 14 at 1000 ppm of Phenobarbital (PB) [14] were mapped onto EC using the above projections. The SUM aggregated expression data were used to calculate the fold changes in expression relative to time-matched controls. The numerical fold changes were transformed into binary (up/down-regulation) for our ILP experiments.

The final step was to combine the metabolomic data with the above integrated Ondex model. We exported the constructed Ondex model into Prolog using the Ondex Prolog export utility. The metabolomic data were manually encoded in Prolog and combined with the Ondex Prolog code. A part of our Ondex Prolog model is shown in Appendix 2. The metabolite changes (increase or decrease) at day 14 at 1000 ppm of PB [12] were combined with the exported Ondex Prolog.

In the Predictive Toxicology application, a new filter was designed for generating variations of background knowledge. This is called a *minimum spanning tree filter* (MiST) in which the projected expression data at the EC level are treated as weights. Intuitively the filter keeps informative reactions in the pathway and filters non-informative reactions out of the pathway. The notion of the informative reaction is based on the level of the projected gene expressions. If a member of EC, *ec*, is associated with a strongly expressed gene, we treat the reaction catalysed by *ec* as informative regardless of up-regulation or down-regulation. We implement this idea by defining the weight as $w = e^{|1/log(fc)|}$ in which fc is a numerical fold change value. Figure 4 shows the MiST algorithm which is based on Kruskal's algorithm [8].

We created 20 variations of background knowledge by applying the Ondex *relation neighbours filter* (RNF) followed by the MiST filter. More precisely, after parsing the KEGG rat database in step 1, we applied the RNF filter to the parsed Ondex model for each neighbourhood distance $k = 1, ..., 10$. The resulting 10 variations of background knowledge are referred to as the background knowledge by RNF. In Appendix 3, we show a visualisation of the background knowledge generated using RNF ($k = 1$) in Figure 9 and its meta-level view in Figure 10.

```
Algorithm MiST
Input: observed metabolites, weighted metabolic pathways.
Output: a subset of the weighted metabolic pathways, OutPath
begin
   1. For each pair of the observed metabolites (m1,m2)
         • make a set of all the paths P from m1 to m2 in which none of the other
           observed metabolite are involved in.
         • calculate the average weight of a path in P. The path with the minimum
           average weight is selected as the path(m1,m2).
         • put path(m1,m2) in a set E.
   2. Sort the members of E in ascending order and store them in an ascending list L.
   3. Take the first entry of L, l,  and select as an edge of OutPath if the new edge does
      not create any cycle in OutPath.
   4. Delete l from L.
   5. Repeat 3 and 4 until all the observed metabolites are connected in OutPath. If fail
      to connect, exit.
   6. Output OutPath and exit.
end
```

Fig. 4. MiST algorithm in the minimum spanning tree filter

For each of these variations, we performed step 2 and step 3 in order to compute the weights at the EC level. We applied the MiST filter for each of the variation and the resulting 10 variants are referred to as the MiST background knowledge.

The variations of the background knowledge, the integrated examples, and non-ground Prolog rules were given to the abductive ILP system Progol5.0. Note that the same non-ground Prolog rules were applied for the Tomato and Predictive Toxicology experiments. Progol5.0 executed abductive inferences in order to generate hypotheses of *inhibition* on reactions for each variation of the background knowledge.

4.2 Results and Discussion

Tomato Application. Leave-one-out cross validation was used to evaluate the experiments, in which predictive accuracy and running time were computed for variation of the background knowledge. Fig. 5(a) shows that the sizes of k-cn generated by RNF increased sharply when $k \leq 2$ but only have minor changes when $k > 2$; whereas the sizes of k-cn generated by SRNF increase gradually and evenly with increasing values of k. Fig. 5(b) indicates that the running time increases linearly with the size of the background knowledge. Fig. 6 shows that (1) all the experiments get at least default predictive accuracy; (2) maximum accuracy could be achieved with relatively shallow bcakground knowledge (i.e. smaller k); (3) the sizes of background knowledge generated by SRNF at which the predictive accuracy reaches its maximum value s smaller than those generated by RNF for two datasets, and are almost level in the third dataset.

In addition, we define a k-model to be the learning result (inhibition/activation abduced) using k-cn. A k-model will be referred to as *stable* if it is equivalent to a K-model (see step 2 of Table 1) with maximum accuracy. The vertical lines in Fig. 5 show that SRNF generates a smaller size background knowledge with less running time to reach the least k values of which k-model starts to be stable than RNF for all the three datasets ($k \geq 3$ for RNF and $k \geq 7$ for SRNF). In summary, it is possible to find more stable, shallow and fine (or smaller) background knowledge that achieves maximum predictive accuracy with less running time by using SRNF rather than RNF. SRNF also enables us to investigate the finer changes between variation of BK in a controllable way. The null hypothesis set for the experiments has been rejected by these results.

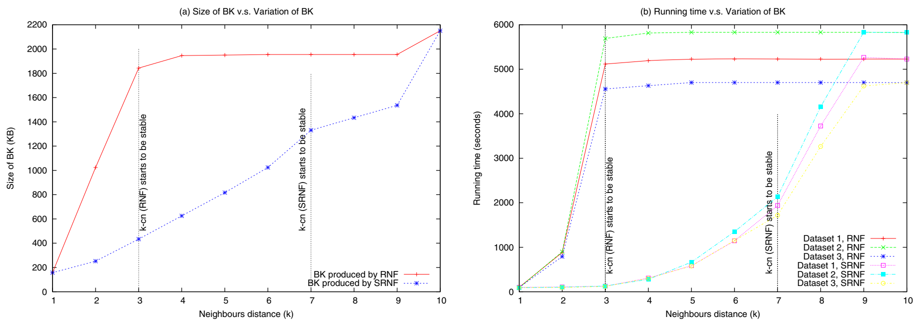


Fig. 5. (a) Size of variation of BK (b) Running time v.s. variation of BK (k-cn) produced by *RNF* and *SRNF* for the three datasets

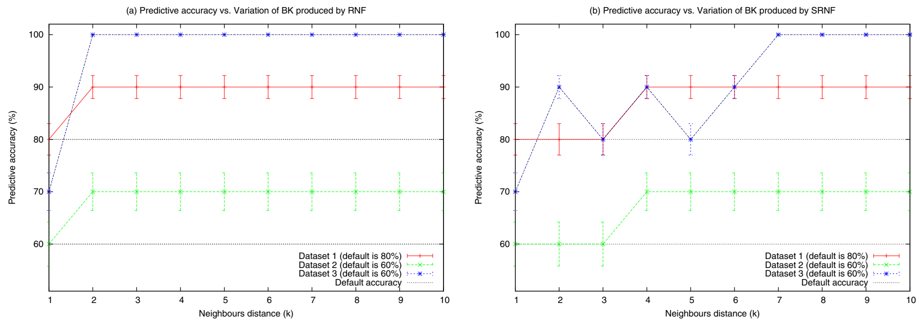


Fig. 6. Predictive accuracy v.s. variation of BK (k-cn) produced by (a) *RNF* and (b) *SRNF* for the three datasets

Predictive Toxicology Application. Leave-one-out cross validation was performed for the selected 9 metabolites. In Figure 7, the sizes of variation of BKs by RNF increased gradually whereas the MiST sizes remained constant. Figure 8 shows that (1) the predictive accuracies of the MiST approach were

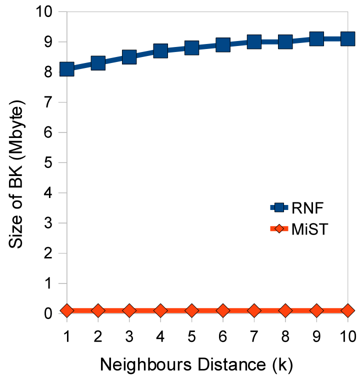


Fig. 7. Size of variations of BKs

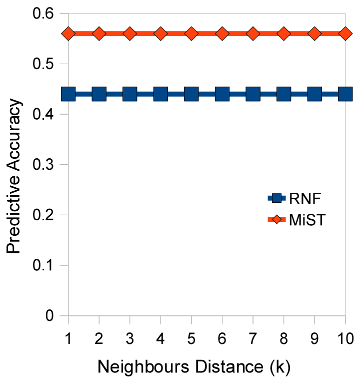


Fig. 8. Predictive accuracy by RNF and MiST

always higher than the results of RNF in which RNF only provided the default accuracy and (2) the best predictive accuracy by MiST was achieved at neighbourhood distance 1.

In the Predictive Toxicology application, the null hypothesis for the experiments has been rejected by these empirical results.

5 Conclusions and Further Work

This paper explores the application of ILP to Systems Biology. These applications involve modelling interations between components of biological systems using abductive inductive logic programing. We also introduce a powerful new system called Ondex which can be used to generate Prolog background knowledge iby parsing and filtering public dataases on cell biochemistry. Two industrial applications are described which are being studied in the Syngenta University Innovation Centre. With the extensive background knowledge generated, we explore the question of how variations of background knowledge affect learning time and predictive accuracy of the same ILP learning system.

Through two independent experiments in tomato biology and predictive toxicology, we conclude that relatively shallow background knowledge can be used to achieve maximum accuracy. In addition the experiments indicate that use of neighbourhod further reduces the learning time required to achieve maximum accuracy.

In further work we aim to improve the non-ground background knowledge rules used in the experiments. We also intend to extend the results using datasets of all time points, and investigate the biological significance of the learned theories.

Acknowledgments

The authors would like to acknowledge the support of Syngenta in its fundig of the University innovations Centre at Imperial College. The first author would also like to thank the Royal Academy of Engineering and Microsoft Research for their support of his research chair.

References

1. Syngenta Ltd., http://www.syngenta.com/en/index.html
2. Syngenta University Innovation Centre, http://www3.imperial.ac.uk/syngenta-uic
3. Chen, J., Muggleton, S.H., Santos, J.: Learning probabilistic logic models from probabilistic examples. Machine Learning 73(1), 55–85 (2008), 10.1007/s10994-008-5076-4
4. Mueller, L.A., et al.: The SOL Genomics Network. A Comparative Resource for Solanaceae Biology and Beyond. Plant Physiology 138(3), 1310–1317 (2005)
5. Goble, C., Stevens, R.: State of the nation in data integration for bioinformatics. Journal of Biomedical Informatics 41(5), 687–693 (2008)
6. Kitano, H.: Computational systems biology. Nature 420, 206–210 (2002)
7. Kohler, J., Baumbach, J., Taubert, J., Specht, M., Skusa, A., Ruegg, A., Rawlings, C., Verrier, P., Philippi, S.: Graph-based analysis and visualization of experimental results with ONDEX. Bioinformatics 22(11), 1383–1390 (2006)
8. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society 7(1), 48–50 (1956)
9. LycoCyc. Solanum lycopersicum database, http://solcyc.solgenomics.net//LYCO/
10. Muggleton, S.H., Bryant, C.H.: Theory completion using inverse entailment. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 130–146. Springer, Heidelberg (2000)
11. Ogata, H., Goto, S., Sato, K., Fujibuchi, W., Bono, H., Kanehisa, M.: KEGG: Kyoto Encyclopedia of Genes and Genomes. Nucl. Acids Res. 27(1), 29–34 (1999)
12. Rubtsov, D.V., Waterman, C., Currie, R.A., Waterfield, C., Salazar, J.D., Wright, J., Griffin, J.L.: Application of a bayesian deconvolution approach for high-resolution 1h nmr spectra to assessing the metabolic effects of acute phenobarbital exposure in liver tissue. Analytical Chemistry 82(11), 4479–4485 (2010)
13. Tamaddoni-Nezhad, A., Chaleil, R., Kakas, A., Muggleton, S.H.: Application of abductive ILP to learning metabolic network inhibition from temporal data. Machine Learning 64, 209–230 (2006), doi:10.1007/s10994-006-8988-x
14. Waterman, C., Currie, R., Cottrell, L., Dow, J., Wright, J., Waterfield, C., Griffin, J.: An integrated functional genomic study of acute phenobarbital exposure in the rat. BMC Genomics 11(1), 9 (2010)

Appendix 1: Sample of projection/2

```
%
%project(KEGG GeneID,  EC_Number).
%
project('RNO:24788_GE','1.1.1.14').
% rno:24788 is mapped onto EC1.1.1.14
project('RNO:24534_GE','1.1.1.27').
% rno:24534 is mapped onto EC1.1.1.27
project('RNO:24533_GE','1.1.1.27').
% rno:24533 is mapped onto EC1.1.1.27
```

Appendix 2: Sample of Ondex Prolog in the Predictive Toxicology domain

```
enzyme('4400','derived enzyme','KEGG','IMPD').
% '4440' is an Ondex ID
concept_name('4400','Nrk1').
% '4400' has the concept name 'Nrkl'
reaction('14366','irreversible','KEGG','IMPD').
% '14366' is an Ondex ID
concept_name('14366','cpd:C05841 => cpd:C05841').
% '14366' for the reaction 'cpd:C05841 => cpd:C05841'
part_of_catalyzing_class('4400','4401','IMPD').
% '4401' is classified into EC '4401'
concept_name('4401','2.7.1.-').
% '2.7.1.-' is an EC
catalyzed_by('14366','4400','IMPD').
% '14366' is catalyzed by '4400'
relation_day14_1500mg(is_related_to,'28501','22436','1.1385442').
% '28501' is an Ondex ID
% '22436' is an Ondex ID
% '1.1385442' is an expression level
probe('28501','UC','IMPD').
% '28501' is for a probe
concept_name('28501','1396933_s_at').
% '1396933_s_at' is a probe name
gene('22436','UC','IMPD').
% '22436' is for a gene
concept_name('22436','RNO:191574_GE').
% 'RNO:191574_GE' is a gene name
```

Appendix 3: Sample of Ondex Models

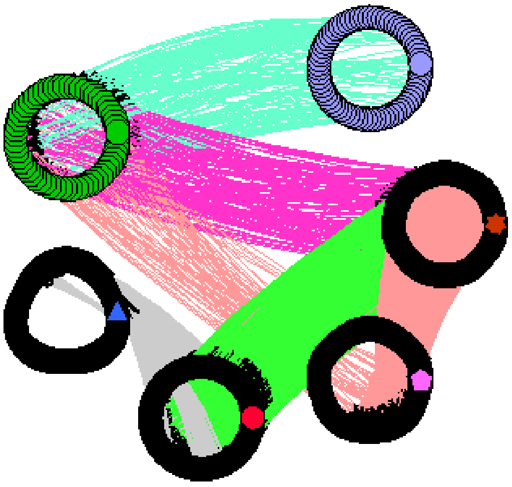


Fig. 9. Visualised concepts in the Predictive Toxicology domain by Ondex (neighbour distance $k = 1$). Each node represents a concept and a directed edge represents a binary relation between two concepts. Users can edit the constructed model by (1) clicking the object in the view and (2) applying Ondex utilities such as filters and relation collapsers.

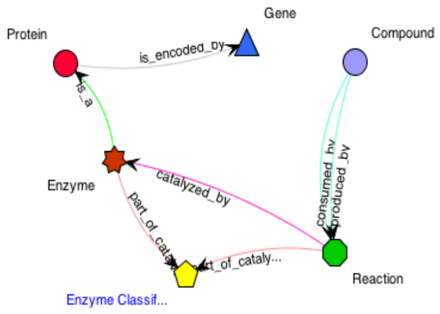


Fig. 10. Meta View of the Ondex model in Figure 9. Users can edit models also at this meta level.

Pruning Search Space for Weighted First Order Horn Clause Satisfiability

Naveen Nair[1,2,3], Anandraj Govindan[2], Chander Jayaraman[2], Kiran TVS[2], and Ganesh Ramakrishnan[2,1]

[1] IITB-Monash Research Academy, Old CSE Building, IIT Bombay
[2] Department of Computer Science and Engineering, IIT Bombay
[3] Faculty of Information Technology, Monash University
{naveennair,ganesh}@cse.iitb.ac.in
{anand.itsme,chanderjayaraman,t.kiran05}@gmail.com

Abstract. Many SRL models pose logical inference as weighted satisfiability solving. Performing logical inference after completely grounding clauses with all possible constants is computationally expensive and approaches such as LazySAT [8] utilize the sparseness of the domain to deal with this. Here, we investigate the efficiency of restricting the Knowledge Base (Σ) to the set of first order horn clauses. We propose an algorithm that prunes the search space for satisfiability in horn clauses and prove that the optimal solution is guaranteed to exist in the pruned space. The approach finds a model, if it exists, in polynomial time; otherwise it finds an interpretation that is most likely given the weights. We provide experimental evidence that our approach reduces the size of search space substantially.

Keywords: First Order Logic, Horn Clauses, MaxSAT, Satisfiability.

1 Introduction

Representing sets of objects and their relationships in a compact form has been the focus of researchers for more than a decade. Weighted first order formulas proved to be one such representation which also allows inference and learning in a structured way. Inference in these sets of formulas are mostly done by Satisfiability testing. We summarize some of the works that addressed the problem of satisfiability in the next paragraph.

Traditional SAT solvers in propositional logic try to find an assignment for all the literals that makes all the clauses true. They return a model if it exists or return unsatisfiable. SAT solvers such as DPLL [6] give exact solutions but employ backtracking and take exponential time in the worst case. Local search methods for satisfying the maximum number of clauses (Max-SAT) has been implemented in GSAT [1], WalkSAT [2] etc. Weighted Max-SAT problems assign weights to the clauses and aim to minimize the sum of the weights of unsatisfied clauses. Teresa et. al. proposed a Lazy approach [9] which uses some bound computation and variable selection heuristic for satisfiability. MiniMaxSAT [4] uses a depth-first branch-and-bound search approach for satisfiability. Satisfiability of first order logic (universally quantified Conjunctive Normal Form (CNF)) can be done by grounding all the clauses (exponential memory cost) and then running satisfiability as in propositional case. Since, many learning techniques require the repeated use of inference and satisfiability, complete grounding of the clauses becomes a bottle neck. Lifted inference [5] techniques used first order variable elimination for probabilistic inference but have not proved their applicability in large domains. As there could be many contradictions in real world, it is better to perform weighted satisfiability. Weighted satisfiability solvers are used for MPE/MAP inference in relational domains [7]. But the complete grounding issue remained unsolved. A LazySAT approach [8] that doesn't ground all clauses was proposed for satisfiability in domains where majority of ground atoms are false. Their approach, a variation of WalkSAT, keeps track of all the clauses that can be affected when a literal in an unsatisfied clause is flipped. Recently in [10], the ground clauses that are satisfied by the evidence are excluded. The approach, which is depended only on the evidence set, processes each clause independently and does not find the dependent clauses transitively. Mihalkova et. al. cluster query literals and perform inference for cluster representatives [12]. Queries are clustered by computing signatures using a recursive procedure based on adjacent nodes. Inference is performed for each cluster representative by running MaxWalkSAT on corresponding Markov Network constructed recursively. Alen Fern mentions about the applicability of Forward Chaining in horn clauses [11] but has not given any algorithm or proof for doing so. In the case of contradicting clauses, it is not straight forward to do forward chaining. The objective of our work is stated in the next paragraphs.

We address the issue of complete grounding by restricting our domain to first order horn clauses and pruning the search space for satisfiability. Our approach caters to several real world applications that use the horn clausal language.

If a set of horn clauses are fully satisfiable, then a minimal model can be found using T_Σ operator (referred to as the immediate consequence operator T_P in [3]) in polynomial time. However weighted unsatisfiable problems require to find the most likely state based on the weights. We propose an extension to the minimal model approach wherein we find (i) the relevant set of ground horn clauses which has a potential to be part of a contradiction and (ii) an interpretation near to the result. MaxSAT algorithm can be used on this subset of clauses, (optionally) starting from the interpretation returned, to get the most likely state. We also prove that local search for optimality in the pruned space cannot affect the satisfiability of the rest of the clauses. Our experiments show that the approach reduces search space substantially and helps maxSAT to converge in short time.

The paper is organized as follows. Section 2 explains the conventional T_Σ operator and the proposed $Modified\_T_\Sigma$ operator. In section 3, we give an overall procedure for satisfiability and also state and prove our claims. Results are discussed in section 4. We conclude our work in section 5.

2 Satisfiability in Horn Clauses

If any of the atoms in the body part of a horn clause is false, then the clause is satisfied because of its inherent structure of having atmost one positive atom and all others negative. The groundings of a set of first order horn clauses (Σ) with all the constants give a large set in which majority of the atoms are false in real world. This makes a large subset of these clauses satisfied by default. We can neglect these clauses and restrict our attention to the clauses that have a potential to be part of a contradiction. We call this set, the relevant set (RS).

We propose an algorithm, $Modified\_T_\Sigma$, to identify the relevant set along with the truth assignments that are almost near to the result. Local search for optimality can be done on this set, starting with the interpretation returned, rather than considering the huge set of clauses and arbitrary truth assignments. Next we explain T_Σ before going to the Modified version.

2.1 T_Σ Operator

T_Σ Operator provides a procedure to generate an interpretation from another. It builds on the concept that for satisfiability in horn clauses, all the unit clauses should be True and if the body of a clause is True, then the head should also be True. Let I_k be the interpretation at the k^{th} step of the operation. Then,

$$I_{k+1} = I_k \cup T_\Sigma(I_k) \tag{1}$$

where, $$T_\Sigma(I) = \{a : a \leftarrow body \in \Sigma \quad and \quad body \subseteq I\} \tag{2}$$

If we start with $I = \emptyset$, and iteratively apply the above function assignment (with respect to the set of clauses), we will eventually converge at an interpretation that is the minimal model of the formulae if one exists. If there is no model for this set, the operation will reach a contradiction and will return *Unsatisfiable.*

In weighted satisfiability problems, if the given set is unsatisfiable, we need to get a most likely state based on the weights. MaxSAT algorithms can do this optimization. Since applying MaxSAT to the complete groundings is expensive, we improve the above method to prune the search space for MaxSAT. $Modified\_T_\Sigma$ Step described in the next section helps us to prune the search space.

2.2 $Modified\_T_\Sigma$ Step

$Modified\_T_\Sigma$ operation returns a model if one exists; Otherwise returns the set of clauses to be used by a local search algorithm and an initial interpretation for the local search. The method is given in Algorithm 1 and is explained below.

Start with applying T_Σ to the set of ground clauses until it converges in a model or some contradiction is attained. In the former case, we can stop and return the current interpretation as the solution. If we land up in a contradiction, we get an atom whose truth value determines the set of clauses satisfied. We assign true to the atom and proceed further till no more clauses can be visited. All the clauses discovered by $Modified\_T_\Sigma$ irrespective of whether satisfied or

Algorithm 1. $Modified\_T_{\Sigma}(\Sigma, DB)$

Input: Σ, the set of first order clauses with weights; DB, evidence set given.
Output: RS, the set of clauses to be considered for optimization; TS, truth assignments of all atoms in RS except those in DB.

```
1. $TS := \emptyset$
2. $RS := \emptyset$
3. for each unit clause $c$ in $\Sigma$ do
4.   for each grounding $c'$ of $c$ do
5.     if $c' \notin RS$ then
6.       Add $c'$ to $RS$
7.     end if
8.     if $c'.head \notin \{TS \cup DB\}$ then
9.       Add $c'.head$ to $TS$
10.    end if
11.  end for
12. end for
13. repeat
14.   for each non unit clause $c$ in $\Sigma$ do
15.     for each grounding $c'$ of $c$ where $c'.body \subseteq \{TS \cup DB\}$ do
16.       if $c' \notin RS$ then
17.         Add $c'$ to $RS$
18.       end if
19.       if $c'.head \notin \{TS \cup DB\}$ then
20.         Add $c'.head$ to $TS$
21.       end if
22.     end for
23.   end for
24. Until no new clauses are added to the set $RS$
25. Return $\{RS, TS\}$
```

not form the relevant set. The interpretation got at the end of the algorithm can optionally be used as initial truth assignment for the optimization step. Note that the truth values for evidences given are always true and cannot be changed.

Any weighted satisfiability algorithm can be applied on the Relevant Set of clauses and the (optional) initial truth values to get a minimum cost interpretation. We now discuss weighted satisfiability approach using $Modified\_T_{\Sigma}$.

3 Modified_Weighted_SAT

In the new approach, $Modified\_T_{\Sigma}$ operation is used to find relevant subset as well as initial truth assignment. Then weighted MaxSAT version given in Algorithm 2 is used. Algorithm 3 gives the overall algorithm.

Claim 1. *All the unsatisfied clauses will be in RS.*

Proof. A horn clause c' is unsatisfied if $c'.body \subseteq \{TS \cup DB\}$ and $c'.head \notin \{TS \cup DB\}$. Step 6 in $Modified\_T_\Sigma$ adds all clauses c' of the form $(c'.head \vee \neg True)$ to RS irrespective of whether it is satisfied or not. Step 17 in $Modified\_T_\Sigma$ adds all clauses c' where $c'.body \subseteq \{TS \cup DB\}$. This covers both the cases of $c'.head$ is $True$ and $c'.head$ is $False$. All other clauses c'' where $c''.body \nsubseteq \{TS \cup DB\}$ are satisfied by default. So set of unsatisfied clauses is a subset of RS. □

Claim 2. *Any flip done in any maxSAT step to make an unsatisfied clause satisfiable only affects the satisfiability of clauses in RS.*

Proof. Let us prove this by contradiction.
Suppose a clause, $c' = (l_1 \vee \neg l_2 \vee \neg l_3 \vee \ldots \vee \neg l_n)$ is not satisfied by the current assignments in $\{TS \cup DB\}$. This happens only when $l_1 \notin \{TS \cup DB\}$ and $\forall i = 2 \ldots n \quad l_i \in \{TS \cup DB\}$. To make c' satisfied, there are two cases. case 1: flip l_1, case 2: flip any of $l_2, l_3, \ldots, l_n$.
case 1: Flip l_1 ($False$ to $True$). Assume that flipping l_1 will affect the state of a clause $c'' \notin RS$. Since $c'' \notin RS$, $c''.body \nsubseteq \{TS \cup DB\}$. Otherwise step 17 in $Modified\_T_\Sigma$ would have covered c'' and it would have been in RS. Also all the unit clauses are covered by step 6 in $Modified\_T_\Sigma$.

Now let $c''.head = l_1$. Since flipping $c''.head$ to $True$ changes the state of c'', $c''.body \subseteq \{TS \cup DB\}$. If this is the case, c'' should have been covered by step 17 in $Modified\_T_\Sigma$ and would have been in RS. Hence the assumption that $c'' \notin RS$ is wrong.

Now let $l_1 \in c''.body$ and flipping it to $True$ changes the state of c''. Then $c''.body \backslash l_1 \subseteq \{TS \cup DB\}$. But applying our approach to c' would have made $l_1 \in TS$ and transitively $c''.body \subseteq \{TS \cup DB\}$ and $c'' \in RS$. Hence the assumption that $c'' \notin RS$ is wrong.
case 2: Flip any $l_i \in \{l_2, l_3, \ldots, l_n\}$ ($True$ to $False$). Assume that flipping l_i will affect the state of a clause $c'' \notin RS$. Since $c'' \notin RS$, $c''.body \nsubseteq \{TS \cup DB\}$. Otherwise step 17 in $Modified\_T_\Sigma$ would have covered c'' and it would have been in RS. Also all the unit clauses are covered by step 6 in $Modified\_T_\Sigma$.

Now let $c''.head = l_i$. Since flipping $c''.head$ to $False$ changes the state of c'', $c''.body \subseteq \{TS \cup DB\}$. If this is the case, c'' should have been covered by step 17 in $Modified\_T_\Sigma$ and would have been in RS. Hence the assumption that $c'' \notin RS$ is wrong.

Now let $l_i \in c''.body$ and flipping it to $False$ changes the sate of c''. Then before flipping, $c''.body \subseteq \{TS \cup DB\}$ which must have been covered by step 17 in $Modified\_T_\Sigma$ and $c'' \in RS$. Hence the assumption that $c'' \notin RS$ is wrong. □

Claim 3. *If α is the cost of an optimal solution to RS, then α is the cost of an optimal solution to Σ*

Proof. let β and γ be the cost of optimal solutions to Σ and RS respectively. That is β should be the sum of costs of RS and $\Sigma \setminus RS$. Increase in cost occurs only because of contradictions and this is in the set RS (proved in claim 1). The best possible solution to the non contradicting part is zero. We get a minimum cost solution for RS part using MaxSAT and any modification to that can result (proved in claim 2 that this doesn't affect $\Sigma \setminus RS$) an increase in cost only in RS. Therefore $\beta = 0 + \gamma$ and thus $\beta = \gamma$ □

Algorithm 2. Modified_Weighted_MaxSAT(Σ_g, TS, DB, $target$)

Input: Σ_g, all grounded clauses with weights; TS, initial truth assignment; DB, the evidence given; target, the upper bound of cost.

Output: TS, An interpretation that is the best solution found.

```
1. $atms$ := atoms in $\Sigma_g$
2. repeat
3.    $cost$ := sum of weights of unsatisfied clauses
4.    if $cost \leq target$
5.       Return Success, $TS$
6.    end if
7.    $c$ := a randomly chosen unsatisfied clause
8.    for each atom $a \in c$ and $a \notin DB$ do
9.       compute DeltaCost($a$), the cost incurred if $a$ is flipped
10.   end for
11.   $a_f$ := $a$ with lowest DeltaCost($a$)
12.   $TS$ := $TS$ with $a_f$ flipped
13.   $cost$ := $cost$ + DeltaCost($a_f$)
14.until the $cost$ is no more decreasing
15.Return Failure, $TS$
```

Algorithm 3. Weighted_HornSAT(Σ, DB, $target$)

Input: Σ, the set of first order clauses with weights; DB, evidence set given; target, maximum cost expected for the optimization step if required.

Output: TS, An interpretation when combined with DB gives the (local) optimum solution.

```
1. $\{RS, TS\}$ := $Modified\_T_\Sigma(\Sigma, DB)$
2. if $\{TS \cup DB\}$ is a model for $\Sigma$ then
3.    Return $TS$
4. else
5.    $TS$ := Modified_Weighted_MaxSAT($RS$, $TS$, $DB$, $target$ )
6. end if
7. Return $TS$
```

4 Results

We implemented new HornSAT algorithm with $Modified\_T_\Sigma$ in java. We have done our experiments in AMD Athlon 64 bit dual core machine (2.90 GHz) with 2.8 GB RAM and running Ubuntu 8.04.

Our results show that, for satisfiablility, the $Modified\_T_\Sigma$ method gives a fewer number of groundings to optimize and that the optimization step converges in a short time when the search space is pruned.

Table 1. Results with uwcse KB and different evidence sets

	No. of groundings		Converged Cost			Time taken (ms)		
Evidence set	Complete	Pruned	Complete grounding + MaxWalkSAT	Pruned + MaxWalkSAT	HornSAT	Complete grounding + MaxWalkSAT	Pruned + MaxWalkSAT	HornSAT
language 181 atoms	508788	6908	90.452	70.265	70.265	2475736	1823778	6896
language 87 atoms	177738	3205	81.463	37.892	37.892	2329459	1098285	2351
AI_766 atoms	Memory Error	182690	NA	344.584	344.584	NA	7507578	7462967

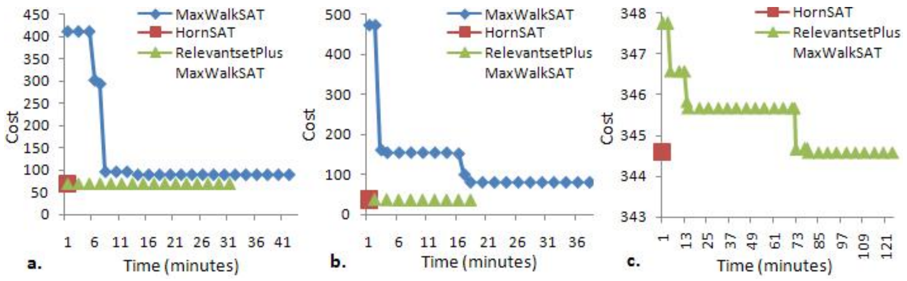


Fig. 1. Comparison of the approaches when applied to uwcse KB. **a.** All 181 atoms from language dataset are given. **b.** 87 atoms from language dataset are given. **c.** All 766 atoms from AI dataset are given. In this experiment, complete grounding approach failed and didnot give any result.

We used the uwcse knowledge base and dataset provided by alchemy [13] for our experiments after making small modifications to make the clause set horn. The constants given as the evidence set is considered as the complete domain for each variables. We have run three experiments on each dataset. First experiment does the complete groundings and runs MaxWalkSAT. Second grounds the clauses with pruning and runs traditional MaxWalkSAT with random truth assignments. The third experiment runs MaxWalkSAT on the pruned clauses set with the initial truth assignment returned by $Modified\_T_{\Sigma}$. Evidence set of different sizes are used and the comparison is given in Table 1. Figures 1.a, 1.b, 1.c portrays the results when 181 atoms of uwcse language dataset, 87 atoms of uwcse language dataset and 766 atoms of uwcse AI dataset are used respectively as evidence set. Experimental results show that the proposed method outperforms the traditional approach in terms of memory and speed. Implementation and other details are available at www.cse.iitb.ac.in/~naveennair/HornSatisfiability/

5 Conclusion and Future Work

Several ground clauses formed as a result of propositionalization of first order horn formulae are satisfied by default and it is a wastage of resources to consider them for optimization. We presented an algorithm that prunes the search space and proved that the optimal solution must lie in the pruned space. Experiments indicate the scope for efficient inference using MaxSAT for the set of horn clauses.

The algorithm can be extended for general clauses by assigning true value artificially to all non-negated atoms if all the negated atoms are true and proceeding like in Algorithm 1.

Acknowledgments. We would like to thank Dr. Ashwin Srinivasan, IBM India Research Laboratory for his helpful comments.

References

1. Selman, B., Levesque, H., Mitchell, D.: A New Method for Solving Hard Satisfiability Problems. In: AAAI 1992, San Jose, CA, pp. 440–446 (1992)
2. Selman, B., Kautz, H., Cohen, B.: Local Search Strategies for Satisfiability Testing. In: Second DIMACS Implementation Challenge on Cliques, Coloring and Satisfiability (1993)
3. Hogger, C.J.: Essentials of logic programming. Oxford University Press, New York (1990)
4. Heras, F., Larrosa, J., Oliveras, A.: MINIMAXSAT: an efficient weighted max-SAT solver. Journal of Artificial Intelligence Research 31(1), 1–32 (2008)
5. Kisynski, J., Poole, D.: Lifted aggregation in directed first-order probabilistic models. In: Proceedings of the 21st International Joint Conference on Artifical Intelligence, California, USA, pp. 1922–1929 (2009)
6. Davis, M., Putnam, H., Logemann, G., Loveland, D.W.: A Machine Program for Theorem Proving. Communications of the ACM 5(7), 394–397 (1962)
7. Singla, P., Domingos, P.: Discriminative training of Markov Logic Networks. In: AAAI 2005, pp. 868–873 (2005)
8. Singla, P., Domingos, P.: Memory-Efficient Inference in Relational Domains. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence, pp. 488–493. AAAI Press, Boston (2006)
9. Alsinet, T., Many, F., Planes, J.: An efficient solver for weighted Max-SAT. Journal of Global Optimization, 61–73 (2008)
10. Shavlik, J., Natarajan, S.: Speeding up inference in Markov logic networks by preprocessing to reduce the size of the resulting grounded network. In: Proceedings of the 21st International JCAI (2009)
11. Fern, A.: A Penalty-Logic Simple-Transition Model for Structured Sequences. In: Computational Intelligence, pp. 302–334 (2009)
12. Mihalkova, L., Richardson, M.: Speeding up inference in statistical relational learning by clustering similar query literals. In: De Raedt, L. (ed.) ILP 2009. LNCS, vol. 5989, pp. 110–122. Springer, Heidelberg (2010)
13. http://alchemy.cs.washington.edu/

Glossary

Atoms: They are predicates in pure form, for *eg: parent(ann,mary)*, *female(X).*

Body: Right side of (:-) (`if`) is called body of the clause.

Clause: Disjunction of literals for *eg:* (parent(ann,mary) $\vee$ $\neg$ female(ann)) $\equiv$ (parent(ann,mary) :- female(ann)).

Clause Representation: Any clause can be written in the form of

`Comma separated positive literals` :- `Comma separated negated literals`,

where (:-) is pronounced as `if`. For example in propositional logic the clause $a \vee b \vee \neg c \equiv$ *a,b* `:-` *c.*

Conjunctive Normal Form (CNF)**:** Every formulae in propositional logic or first-order logic is equivalent to a formula that can be written as a conjunction of disjunctions i.e, something like $(a\,(X) \vee b\,(X)) \wedge (c\,(X) \vee d\,(X)) \wedge \cdots$. When written in this way the formula is said to be in conjunctive normal form or CNF.

Constants: A constant symbol represents an individual in the world. In first order logic it is represented by small letter *eg: jane*, *1*, *a etc.*

Definite Clause: Clauses with exactly one positive literal *eg: p(X) :- c(X),d(Y).*

Facts: Body less horn clauses, for *eg: female(ann)*; *daughter(mary).*

Functions: Take input as tuple of objects and return another object *eg: motherOf(ann)*, *parentOf(mary).*

Ground Clause: Clauses formed as a result of replacing each variable by all possible constants in each predicate of a clause.

Head: Left side of (:-) (*if*) is called head of the clause.

Herbrand Interpretation: A (Herbrand) interpretation is a truth assignment to all the atoms formed as a result of replacing the variables in a predicate by all the possible constants (objects).

Herbrand Model: a Herbrand model is simply a Herbrand interpretation that makes a wellformed formula true.

Horn Clause: Clause with atmost one positive literal for *eg: (:- parent(ann,joan), female(joan).)* , *(parent(ann,kan) :- female(mary).)*

In a horn clause in CNF, all the atoms preceeded by a ¬ form the body part and the atom not preceded by a ¬ is the head. Here ¬A means (not)A.

Knowledge Base: A Knowledge Base is a set of clauses which represents a theory.

Literals: They are predicates in either pure form or negated form, for *eg: ¬parent(ann,mary).*

MaxSAT: It is a local search method used for satisfying the maximum number of clauses, which starts with random truth assignments to all ground atoms and improve the solution step by step by flipping one literal at a time which makes some clauses satisfied and some others unsatisfied.

Model: An interpretation which makes the clause true. For eg: $P : -Q, R$, the models are $M = \phi, \{P, Q, R\}$.

Statistical Relational Learning (SRL): Statistical relational learning deals with machine learning and data mining in relational domains where observations may be missing, partially observed, and/or noisy.

Variables: Starts with the capital letters for *eg: X,*Abs, *etc.*

Weighted MaxSAT: Given a first order Knowledge Base (σ), find a Herbrand Interpretation that maximizes (or minimizes) the sum of weights of satisfied (unsatisfied) clauses.

Multi-relational Pattern Mining Based-on Combination of Properties with Preserving Their Structure in Examples

Yusuke Nakano and Nobuhiro Inuzuka

Nagoya Institute of Technology,
Gokiso-cho, Showa, Nagoya 466-8555, Japan
Tel.: +81-52-735-5050, Fax: +81-52-735-5473
nakano18115115@gmail.com, inuzuka@nitech.ac.jp

Abstract. We propose an algorithm for multi-relational pattern mining through the problem established in WARMR. In order to overcome the combinatorial problem of large pattern space, another algorithm MAPIX restricts patterns into combination of basic patterns, called properties. A property is defined as a set of literals appeared in examples and is of an extended attribute-value form. Advantage of MAPIX is to make patterns from pattern fragments occurred in examples. Many patterns which are not appeared in examples are not tested. Although the range of patterns is clear and MAPIX enumerates them efficiently, a large part of patterns are out of the range. The proposing algorithm keeps the advantage and extends the way of combination of properties. The algorithm combines properties as they appeared in examples, we call it structure preserving combination.

1 Introduction

This paper studies multi-relational pattern mining obeying the line of Warmr [2,3]. Warmr generates and tests candidate patterns with a pruning technique similar to Apriori [1]. In spite of the cut-down procedure it has the exponentially growing space of hypothesis with respect to the length of patterns and the number of relations. Another algorithm Mapix restricts patterns into conjunctions of basic patterns, called properties. A property is an extended attribute-value form consisting of literals that refer to parts of objects and a literal that describes a property of or a relation among the parts. For example for a person (or his/her family) a basic pattern consists of referential literals (or an extended attribute), for example "one of his/her grandchildren", and a descriptive literal (or an extended attribute value), for examples "it is male" and then it describes a basic property, for example "this person has a grandson".

Mapix finds patterns made of properties appeared in samples relying on type and mode information of relations. The search can be seen as a combination of bottom-up and top-down search, it constructs properties from samples in a bottom-up way and tests patterns combining the properties in a top-down way.

The properties of Mapix and their conjunctions are natural but the range in which Mapix generates patterns is still narrow. There seems another natural range of patterns to be generated. We propose another way to combine properties. For example, for a person of a family, “having a son” and “having a granddaughter” are properties and by combining them we may have “having a son and a granddaughter”. But it may not represent a real situation of the family precisely, in which it may happen that “having a son who has a daughter”.

We propose an algorithm by using a structural combination [4] of basic patterns. We propose to use preserved structure in a sample and give a simple algorithm. Our algorithm generates patterns in a larger pattern space.

2 Patterns and Mapix Algorithm

Datalog is used to represent data and patterns. Datalog clauses are of the form $\forall(h \leftarrow b_1 \wedge \ldots \wedge b_n)$ without functors, where $\forall F$ means all variables in F are universally quantified and $\forall$ is omitted when understood. For $c = h \leftarrow b_1 \wedge \ldots \wedge b_n$, head($c$) denotes the head atom h and body(c) denotes the body conjunction $b_1 \wedge \ldots \wedge b_n$. A fact is a clause without body. A substitution is described by $\theta = \{v_1/t_1, \ldots, v_n/t_n\}$ for variables v_i and terms t_i. $P\theta$ for a formula P means replacing every variable v_i with t_i.

For our mining task a Datalog DB $\boldsymbol{R}$ is given and one of extensional relations is specified for a *target* (It corresponds to the concept *key* of Warmr). A fact of the target relation is called a *target instance*.

A *query* is a clause without head $\leftarrow b_1 \wedge \ldots \wedge b_n$, equivalently an existentially quantified conjunction $\exists(b_1 \wedge \ldots \wedge b_n)$, where $\exists Q$ means all variables in Q are existentially quantified. When a formula is clearly meant to be a query $\exists$ is dropped. A query q is said to succeed wrt $\boldsymbol{R}$ when $\boldsymbol{R} \models \exists q$.

The following gives patterns, among which we are interested in frequent ones.

Definition 1 (pattern). *A pattern is a Datalog clause whose head is of the target predicate. For a target instance e and a pattern P, $P(e)$ denotes a query $\exists(body(P)\theta)$ where θ is the mgu (most general unifier) of e and head(P). When $P(e)$ succeeds we say that e* possesses *P.*

Definition 2 (frequent pattern). *The* frequency *of P is the number of target instances which possess P. P is* frequent *if its frequency exceeds $sup_{min} \cdot N$, where sup_{min} is a given minimal support and N is the number of all target instances.*

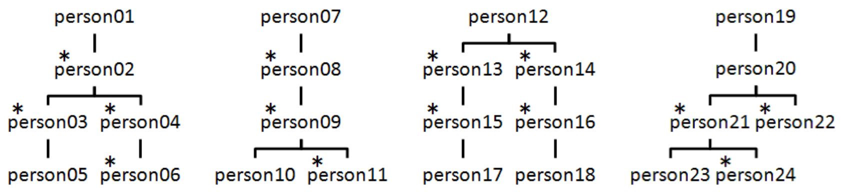
Example 1 (running example). Let us consider a DB $\boldsymbol{R}_{\mathbf{fam}}$ on families (Fig. 1). It includes four relations, $\mathsf{parent}(x, y)$ meaning x is a parent of y, $\mathsf{female}(x)$ for a female x, $\mathsf{male}(x)$ for a male x, and $\mathsf{grandfather}(x)$ meaning x is someone's grandfather. We use gf, p, m, f for the relations for short. We also abbreviate `person01` as $\mathsf{01}$. Let gf be a target.

Then, for example the following formula is a pattern.

$$P = \mathsf{gf}(A) \leftarrow \mathsf{m}(A) \wedge \mathsf{p}(A, B) \wedge \mathsf{f}(B)$$

For a target instance $\epsilon = \mathsf{gf}(\mathsf{01})$, $P(\epsilon)$ denotes a query,

$$P(\epsilon) = \exists((\mathsf{m}(A) \wedge \mathsf{p}(A, B) \wedge \mathsf{f}(B))\theta) = \exists(\mathsf{m}(\mathsf{01}) \wedge \mathsf{p}(\mathsf{01}, B) \wedge \mathsf{f}(B)).$$



grandfather
person01
person07
person12
person19
person20

parent	
person01	person02
person02	person03
person02	person04
person03	person05
...	...

male
person01
person05
person07
person10
...

female
person02
person03
person04
person06
...

Fig. 1. The family DB $R_{\mathbf{fam}}$, including grandfather, parent, male and female, of which grandfather is a target. The persons with $*$ are female and others are male.

where θ is the mgu of ϵ and head(P). The query $P(\epsilon)$ succeeds by an assignment $\{B \mapsto 02\}$ then e possesses P. □

Many ILP algorithms assume modes for predicates to restrict patterns. Some arguments of a predicate have a role as input (denoted by $+$) and some as output ($-$). We give parent$(+,-)$, male$(+)$, and female$(+)$ to the predicates in $R_{\mathbf{fam}}$.

We distinguish between two classes of predicates. Predicates with at least one $\langle - \rangle$-arg. are called *path predicates*, e.g. parent$(+,-)$, which have a role like a function generating a term from others. Predicates without $\langle - \rangle$-arg. are called *check predicates*, e.g. male$(+)$ and female$(+)$, which have a role describing a property of given terms. An instance of a path/check predicate in DB is called a path/check literal. We do not give mode for target predicate.

Using these concepts Mapix extracts basic patterns, called *properties*, from DB and generalize them to basic patterns, called *property items*.

Definition 3 (property). *A* property *of a target instance e wrt DB $\boldsymbol{R}$ is a minimal set L of ground atoms in $\boldsymbol{R}$ satisfying*

1. *L includes exactly one check literal, and*
2. *L can be given a linear order where every term in a $\langle + \rangle$-arg. of a literal in L is occurred in some precedent literals in the order or e.*

Definition 4 (variablization). *For a ground formula α a formula β is a* variablization *of α when*

1. *β does not include any ground term, and*
2. *there exists a substitution $\theta = \{v_1/t_1, \cdots, v_n/t_n\}$ that satisfies (a) $\alpha = \beta\theta$ and (b) $t_1, \ldots, t_n$ in θ are all different terms in α.*

We assume to use new variables never used before when variablizing.

Definition 5. *For a set $L = \{l_1, \cdots, l_m\}$ of ground literals and a target instance e $\mathbf{var}(e \leftarrow L)$ denotes a variablization of $e \leftarrow l_1 \wedge \ldots \wedge l_m$.*

Table 1. Properties and property items of $\epsilon = \mathsf{gf}(\mathsf{01})$

$\mathsf{pr0}=\{\mathsf{m(01)}\}$	$\mathsf{it0}=\mathsf{gf}(A)\leftarrow\mathsf{m}(A)$
$\mathsf{pr1}=\{\mathsf{p(01,02)},\mathsf{f(02)}\}$	$\mathsf{it1}=\mathsf{gf}(A)\leftarrow\mathsf{p}(A,B)\wedge\mathsf{f}(B)$
$\mathsf{pr2}=\{\mathsf{p(01,02)},\mathsf{p(02,03)},\mathsf{f(03)}\}$ ($\{\mathsf{p(01,02)},\mathsf{p(02,04)},\mathsf{f(04)}\}$)	$\mathsf{it2}=\mathsf{gf}(A)\leftarrow\mathsf{p}(A,B)\wedge\mathsf{p}(B,C)\wedge\mathsf{f}(C)$
$\mathsf{pr3}=\{\mathsf{p(01,02)},\mathsf{p(02,03)},\mathsf{p(03,05)},\mathsf{m(05)}\}$	$\mathsf{it3}=\mathsf{gf}(A)\leftarrow\mathsf{p}(A,B)\wedge\mathsf{p}(B,C)\wedge\mathsf{p}(B,D)\wedge\mathsf{m}(D)$
$\mathsf{pr4}=\{\mathsf{p(01,02)},\mathsf{p(02,04)},\mathsf{p(04,06)},\mathsf{f(06)}\}$	$\mathsf{it4}=\mathsf{gf}(A)\leftarrow\mathsf{p}(A,B)\wedge\mathsf{p}(B,C)\wedge\mathsf{p}(C,D)\wedge\mathsf{f}(D)$

When L is a property of e, $\mathbf{var}(e \leftarrow L)$ is called a *property item* of e. Possessing P by e and a query $P(e)$ are defined as in Definition 1.

Example 2. $L=\{\mathsf{p(01,02)},\mathsf{p(02,03)},\mathsf{f(03)}\}$ is a property of $\epsilon=\mathsf{gf(01)}$. Then $\mathsf{it} = \mathbf{var}(\epsilon\leftarrow L)=\mathsf{gf}(A)\leftarrow\mathsf{p}(A,B)\wedge\mathsf{p}(B,C)\wedge\mathsf{f}(C)$ is possessed by ϵ, i.e. $\boldsymbol{R}_{\mathbf{fam}} \models \mathsf{it}(\epsilon)$. □

Then, Mapix algorithm is as follows:

1. It samples an appropriate number of target instances from a target relation.
2. For each sampled instance it extracts property items hold on DB.
3. It executes an Apriori-like level-wise frequent pattern mining algorithm by regarding the satisfaction of a property item as possession of it.

As discussed in [5] the size of sampled instances in step 1 can be constant with respect to the size of all examples.

Example 3. Table 1 shows property items produced from $\mathsf{gf(01)}$. Only these are frequent for min sup 60% even if we sample all instances, when another infrequent property items $\mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{p}(B,C) \wedge \mathsf{m}(C)$ is extracted from $\mathsf{gf(20)}$.

In step 3 Mapix combines property items by a simple conjunction. For example $\mathsf{it2}$ (means having a granddaughter) and $\mathsf{it4}$ (means having a great-granddaughter) are combined to the following pattern, which we write as $\langle\mathsf{it2},\mathsf{it4}\rangle$,

$$\langle\mathsf{it2},\mathsf{it4}\rangle = \mathsf{gf}(A)\leftarrow\mathsf{p}(A,B)\wedge\mathsf{p}(B,C)\wedge\mathsf{f}(C)\wedge\mathsf{p}(A,D)\wedge\mathsf{p}(D,E)\wedge\mathsf{p}(E,F)\wedge\mathsf{f}(F),$$

which means having a granddaughter and a great-granddaughter but neither having a child who has a daughter and a granddaughter nor having a granddaughter who has a daughter. We call such a conjunction a *property itemset.*

3 Ideas and an Algorithm

We propose an algorithm based on Mapix in order to produce rich combinations of properties. It keeps efficiency by seeing only combination appeared in samples.

In order to explain our idea, we introduce the shadow of a property item, which is a set of properties that produce the property item.

Definition 6. *For a database R and a property item* it*, the set defined below is called the shadow of the property item,*

$$\mathbf{shadow}(\mathsf{it},R) = \{\text{`}e \leftarrow L\text{'} \in T \times 2^R \mid L \textit{ is a property and } \mathbf{var}(e \leftarrow L)\sim\mathsf{it}\},$$

where T is the target relation in R, 2^X is the power set of a set X, and $P\sim Q$ means P and Q are θ-equivalent, i.e. P θ-subsumes Q and Q θ-subsumes P.

Example 4. For $R_{\mathbf{fam}}$ and $\mathsf{it2}$, the shadow of it is

$$\begin{aligned}&\mathbf{shadow}(\mathsf{it2}, R_{\mathbf{fam}}) = \{\\ &\mathsf{gf(01)} \leftarrow \{\mathsf{p(01,02)}, \mathsf{p(02,03)}, \mathsf{f(03)}\}, \mathsf{gf(01)} \leftarrow \{\mathsf{p(01,02)}, \mathsf{p(02,04)}, \mathsf{f(04)}\},\\ &\mathsf{gf(07)} \leftarrow \{\mathsf{p(07,08)}, \mathsf{p(08,09)}, \mathsf{f(09)}\}, \mathsf{gf(12)} \leftarrow \{\mathsf{p(12,13)}, \mathsf{p(13,15)}, \mathsf{f(15)}\},\\ &\mathsf{gf(12)} \leftarrow \{\mathsf{p(12,14)}, \mathsf{p(14,16)}, \mathsf{f(16)}\}, \mathsf{gf(19)} \leftarrow \{\mathsf{p(19,20)}, \mathsf{p(20,21)}, \mathsf{f(21)}\},\\ &\mathsf{gf(19)} \leftarrow \{\mathsf{p(19,20)}, \mathsf{p(20,22)}, \mathsf{f(22)}\}, \mathsf{gf(20)} \leftarrow \{\mathsf{p(20,21)}, \mathsf{p(21,24)}, \mathsf{f(24)}\} \;\}\end{aligned}$$

Definition 7 (combinable property itemset). *A property itemset* $\langle \mathsf{it}_{i_1}, \ldots, \mathsf{it}_{i_n} \rangle$ *is* combinable, *if there exists a target instance* e *and*

$$\langle e \leftarrow \mathsf{pr}_{i_1}, \ldots, e \leftarrow \mathsf{pr}_{i_n} \rangle \in \mathbf{shadow}(\mathsf{it}_{i_1}, R) \times \ldots \times \mathbf{shadow}(\mathsf{it}_{i_n}, R),$$

such that $\bigcap_{j=1,\ldots,n}(\mathbf{terms}(\mathsf{pr}_{i_j}) - \mathbf{terms}(e)) \neq \emptyset$*, where* $\mathbf{terms}(p)$ *is the set of all terms in* p*.* $\langle e \leftarrow \mathsf{pr}_{i_1}, \ldots, e \leftarrow \mathsf{pr}_{i_n} \rangle$ *is called a combinable shadow tuple.*

Definition 8 (combined property item). *When a property itemset* $\mathsf{is} = \langle \mathsf{it}_{i_1}, \ldots, \mathsf{it}_{i_n} \rangle$ *is combinable and* $\langle e \leftarrow \mathsf{pr}_{i_1}, \ldots, e \leftarrow \mathsf{pr}_{i_n} \rangle$ *is a combinable shadow tuple of it,*

$$\mathbf{var}(e \leftarrow \textstyle\bigcup_{j=1,\ldots,n} \mathsf{pr}_{i_j})$$

is called a combined property item produced from is*. A property item that is not combined is called* atomic*.*

Example 5. A property itemset $\langle \mathsf{it2}, \mathsf{it4} \rangle$ is combinable, because the shadow of $\mathsf{it2}$ (see the example above) and the shadow of $\mathsf{it4}$

$$\begin{aligned}\mathbf{shadow}(\mathsf{it4}, R_{\mathbf{fam}}) = \{\, &\mathsf{gf(01)} \leftarrow \{\mathsf{p(01,02)}, \mathsf{p(02,04)}, \mathsf{p(04,06)}, \mathsf{f(06)}\},\\ &\mathsf{gf(07)} \leftarrow \{\mathsf{p(07,08)}, \mathsf{p(08,09)}, \mathsf{p(09,11)}, \mathsf{f(11)}\},\\ &\mathsf{gf(19)} \leftarrow \{\mathsf{p(19,20)}, \mathsf{p(20,21)}, \mathsf{p(21,24)}, \mathsf{f(24)}\} \;\}\end{aligned}$$

have five combinable shadow tuples

$$\begin{aligned}&\langle \mathsf{gf(01)} \leftarrow \{\mathsf{p(01,02)}, \mathsf{p(02,03)}, \mathsf{f(03)}\}, \mathsf{gf(01)} \leftarrow \{\mathsf{p(01,02)}, \mathsf{p(02,04)}, \mathsf{p(04,06)}, \mathsf{f(06)}\}\rangle,\\ &\langle \mathsf{gf(01)} \leftarrow \{\mathsf{p(01,02)}, \mathsf{p(02,04)}, \mathsf{f(04)}\}, \mathsf{gf(01)} \leftarrow \{\mathsf{p(01,02)}, \mathsf{p(02,04)}, \mathsf{p(04,06)}, \mathsf{f(06)}\}\rangle,\\ &\langle \mathsf{gf(07)} \leftarrow \{\mathsf{p(07,08)}, \mathsf{p(08,09)}, \mathsf{f(09)}\}, \mathsf{gf(07)} \leftarrow \{\mathsf{p(07,08)}, \mathsf{p(08,09)}, \mathsf{p(09,11)}, \mathsf{f(11)}\}\rangle,\\ &\langle \mathsf{gf(19)} \leftarrow \{\mathsf{p(19,20)}, \mathsf{p(20,21)}, \mathsf{f(21)}\}, \mathsf{gf(19)} \leftarrow \{\mathsf{p(19,20)}, \mathsf{p(20,21)}, \mathsf{p(21,24)}, \mathsf{f(24)}\}\rangle,\\ &\langle \mathsf{gf(19)} \leftarrow \{\mathsf{p(19,20)}, \mathsf{p(20,22)}, \mathsf{f(22)}\}, \mathsf{gf(19)} \leftarrow \{\mathsf{p(19,20)}, \mathsf{p(20,21)}, \mathsf{p(21,24)}, \mathsf{f(24)}\}\rangle\end{aligned}$$

and they produce the following two combined property items.

$$\begin{aligned}\mathsf{it2\text{-}4} &= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{p}(B,C) \wedge \mathsf{f}(C) \wedge \mathsf{p}(C,D) \wedge \mathsf{f}(D).\\ \mathsf{it2\text{-}4'} &= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{p}(B,C) \wedge \mathsf{f}(C) \wedge \mathsf{p}(B,D) \wedge \mathsf{p}(D,E) \wedge \mathsf{f}(E).\end{aligned}$$

$\mathsf{it2\text{-}4}$ means having a granddaughter who has a daughter and $\mathsf{it2\text{-}4'}$ means having a child who has a daughter and a granddaughter. Every tuple produces a combined property item equivalent to one of the above. All frequent combined property items produced from $R_{\mathbf{fam}}$ are shown in Table 2.

Table 2. All frequent property items and combined property items in $R_{\mathbf{fam}}$ for min sup=60%

it0	$= \mathsf{gf}(A) \leftarrow \mathsf{m}(A).$
it1	$= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{f}(B).$
it2	$= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{p}(B,C) \wedge \mathsf{f}(C).$
it3	$= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{p}(B,C) \wedge \mathsf{p}(C,D) \wedge \mathsf{m}(D).$
it4	$= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{p}(B,C) \wedge \mathsf{p}(C,D), \mathsf{f}(D).$
it1-2	$= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{f}(B) \wedge \mathsf{p}(B,C) \wedge \mathsf{f}(C).$
it1-3	$= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{f}(B) \wedge \mathsf{p}(B,C) \wedge \mathsf{p}(C,D) \wedge \mathsf{m}(D).$
it2-3	$= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{p}(B,C) \wedge \mathsf{f}(C) \wedge \mathsf{p}(C,D) \wedge \mathsf{m}(D).$
it2-3'	$= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{p}(B,C) \wedge \mathsf{f}(C) \wedge \mathsf{p}(B,D) \wedge \mathsf{p}(D,E) \wedge \mathsf{m}(E).$
it2-4	$= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{p}(B,C) \wedge \mathsf{f}(C) \wedge \mathsf{p}(C,D) \wedge \mathsf{f}(D).$
it2-4'	$= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{p}(B,C) \wedge \mathsf{f}(C) \wedge \mathsf{p}(B,D) \wedge \mathsf{p}(D,E) \wedge \mathsf{f}(E).$
it3-4	$= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{p}(B,C) \wedge \mathsf{p}(C,D) \wedge \mathsf{f}(D) \wedge \mathsf{p}(B,E) \wedge \mathsf{p}(E,F), \mathsf{m}(F).$
it1-2-3	$= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{f}(B) \wedge \mathsf{p}(B,C) \wedge \mathsf{f}(C) \wedge \mathsf{p}(C,D) \wedge \mathsf{m}(D).$
it1-2-3'	$= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{f}(B) \wedge \mathsf{p}(B,C) \wedge \mathsf{f}(C) \wedge \mathsf{p}(B,D) \wedge \mathsf{p}(D,E) \wedge \mathsf{m}(E).$
it2-3-4	$= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{p}(B,C) \wedge \mathsf{f}(C) \wedge \mathsf{p}(C,D) \wedge \mathsf{f}(D) \wedge \mathsf{p}(B,E) \wedge \mathsf{p}(E,F) \wedge \mathsf{m}(F).$
it2-3-4'	$= \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{p}(B,C) \wedge \mathsf{p}(C,D) \wedge \mathsf{f}(D) \wedge \mathsf{p}(B,E) \wedge \mathsf{f}(E) \wedge \mathsf{p}(E,F) \wedge \mathsf{m}(F).$

With atomic property items combined property items work to make other patterns by conjunction. For example, it1 and it2-4 make the following pattern,

$$\langle \mathsf{it1}, \mathsf{it2\text{-}4} \rangle = \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B) \wedge \mathsf{f}(B) \wedge \mathsf{p}(A,C) \wedge \mathsf{p}(C,D) \wedge \mathsf{f}(D) \wedge \mathsf{p}(D,E) \wedge \mathsf{f}(E),$$

which means having a daughter and a granddaughter who has a daughter. □

Here we propose a new algorithm for all frequent patterns made from conjunction among atomic and combined property items extracted from samples:

1. It samples target instances from a target relation.
2. For each sampled instance it extracts atomic property items hold on DB.
3. By using an Apriori-like level-wise algorithm it enumerates all frequent conjunctions of the atomic property items.
4. It produces all combined property items from the frequent combinable conjunction.
5. Again by the level-wise algorithm it enumerates all frequent conjunctions of atomic and combined property items.

The detail of the algorithm is given in Table 3.

4 Experiments and Concluding Remarks

We have done two experiments. The first one was with $R_{\mathbf{fam}}$, where we aim to see the varieties of patterns extracted. Table 4 shows the numbers of patterns enumerated and runtime for our algorithm as well as MAPIX and the algorithm in [4] according as the minimum support threshold is changed. Our algorithm produced more patterns than others. There was no duplication in the patterns enumerated by three algorithms in the sense of θ-equivalence.

Table 3. A proposing algorithm

```
input   $\boldsymbol{R}$ : a DB; $T$ : target relation; $\text{sup}_{\text{min}}$: the min. sup. threshold;
output  $Freq$: the set of patterns whose supports are larger than $\text{sup}_{\text{min}}$
 1.  Select an appropriate size of subset $T' \subseteq T$;
 2.  $Items := \emptyset$; $\mathcal{P} := \emptyset$; $Freq := \emptyset$;
 3.  For each $e \in T'$ do $\mathcal{P} := \mathcal{P} \cup \{e \leftarrow \mathsf{pr} \mid \mathsf{pr}$ is a property of $e\}$
 4.  For each '$e \leftarrow \mathsf{pr}$' $\in \mathcal{P}$ do
 5.      If $\exists I \in Items$, $I \sim \mathbf{var}(e \leftarrow \mathsf{pr})$ then $S[I] := S[I] \cup \{e \leftarrow \mathsf{pr}\}$;
 6.      else $I' = \mathbf{var}(e \leftarrow \mathsf{pr})$; $S[I'] := \{e \leftarrow \mathsf{pr}\}$ ; $Items := Items \cup \{I'\}$;
 7.  $k := 1$; $\mathcal{F}_1^1 := \{\langle I \rangle \mid I \in Items$ and $\mathbf{supp}(I) \geq \text{sup}_{\text{min}}\}$; $Freq := \mathcal{F}_1^1$;
 8.  While $\mathcal{F}_k^1 \neq \emptyset$ do
 9.      $\mathcal{C}_{k+1} := \text{CANDIDATE}(\mathcal{F}_k^1, \mathcal{F}_k^1)$; $\mathcal{F}_{k+1}^1 := \{IS \in \mathcal{C}_{k+1} \mid \mathbf{supp}(IS) \geq \text{sup}_{\text{min}}\}$;
10.      $Freq := Freq \cup \mathcal{F}_{k+1}^1$; $k := k+1$;
11.  $Combined := \text{CANDICOMB}(Freq)$;
12.  $k := 1$; $\mathcal{F}_1^2 := \{\langle I \rangle \mid I \in Combined$ and $\mathbf{supp}(I) \geq \text{sup}_{\text{min}}\}$; $Freq := Freq \cup \mathcal{F}_1^2$;
13.  While $\mathcal{F}_k^2 \neq \emptyset$ do
14.      $\mathcal{C}_{k+1} := \text{CANDIDATE}(\mathcal{F}_k^1, \mathcal{F}_k^2)$; $\mathcal{F}_{k+1}^2 := \{IS \in \mathcal{C}_{k+1} \mid \mathbf{supp}(IS) \geq \text{sup}_{\text{min}}\}$;
15.      $Freq := Freq \cup \mathcal{F}_{k+1}^2$; $k := k+1$;
16.  Return $Freq$;

CANDIDATE($\mathcal{F}_k^1$,$\mathcal{F}_k^2$):
input   $\mathcal{F}_k^1, \mathcal{F}_k^2$ : sets of frequent property itemsets of a level;
output  $\mathcal{C}_{k+1}$ : the set of candidate property itemsets of the next level where at least
                a property itemset is used from $\mathcal{F}_k^2$;
 1.  $\mathcal{C}_{k+1} := \emptyset$
 2.  For each pair $\langle\langle I_1, \ldots, I_k\rangle, \langle I'_1, \ldots, I'_k\rangle\rangle \in \mathcal{F}_k^2 \times \{\mathcal{F}_k^1 \cup \mathcal{F}_k^2\}$
       where $I_1 = I'_1, \ldots, I_{k-1} = I'_{k-1}$, and $I_k < I'_k$ do
 3.      $\mathcal{C}_{k+1} := \mathcal{C}_{k+1} \cup \{\langle I_1, \ldots, I_{k-1}, I_k, I'_k\rangle\}$;
 4.  For each $IS \in \mathcal{C}_{k+1}$ do
 5.      If $k = 1$ and ($I \preceq I'$ or $I' \preceq I$), where $IS = \langle I, I'\rangle$ then remove $IS$ from $\mathcal{C}_{k+1}$;
 6.      For each $I \in IS$ do if $IS - \{I\} \notin \mathcal{F}_k^1 \cup \mathcal{F}_k^2$ then remove $IS$ from $\mathcal{C}_{k+1}$;
 7.  Return $\mathcal{C}_{k+1}$;

CANDICOMB($Freq$):
input   $Freq$ : the set of frequent property item sets made of atomic items;
output  $Combined$ : the set of combined property items produced from
                      all property itemsets in $Freq$;
 1.  $Combined := \emptyset$;
 2.  For each $\langle I_1, \ldots, I_n\rangle \in Freq$ for $n \geq 2$ do
 3.      For each $\langle e_1 \leftarrow \mathsf{pr}_1, \ldots, e_n \leftarrow \mathsf{pr}_n\rangle \in S[I_1] \times \cdots \times S[I_n]$ do
 4.          If $e_1 = \ldots = e_n$ and $\bigcap_{j=1,\ldots,n}(\mathbf{terms}(\mathsf{pr}_j) - \mathbf{terms}(e_j)) \neq \emptyset$ then
 5.              $Combined := Combined \cup \{\mathbf{var}(e_1 \leftarrow \bigcup_{j=1,\ldots,n} \mathsf{pr}_j)\}$;
 6.  For each $I \in Combined$ do
 7.      If $\exists I' \in Combined$, $I' \neq I \wedge I' \sim I$ then $Combined := Combined - \{I\}$;
 6.  Return $Combined$;
```

The second experiment was with the data of Bongard. Fig. 2 shows the number of patterns and runtime when the number of examples to extract properties changes. These are the average of 10 times execution. We used all examples to count frequency but sampled limited number of examples to extract properties. By 80 examples our algorithm produced the same set of patterns (802 patterns) as the case using whole 392 examples. The 802 patterns had no duplication. Fig. 2 also shows the grow of runtime according to the sample size.

Our algorithm enumerates larger range of patterns, made of property items and combined property items in which property items are combined as in examples. The algorithm in [4] uses structural combination but it has limitation. It treats property items only in a single example then it composes all examples as an artificial large example. Instead of our shadow the algorithm keeps the conjunction of all equivalent property items because the conjunction is equivalent to each property item. By the method patterns becomes larger. Our algorithm overcome these limitation by the shadow and simple algorithm, but has the large time complexity to the size of examples sampled. In fact it took 6556 seconds for Bongard when it samples all examples. The sufficient size of examples, however, can be suppressed because it depends only on the minimum support threshold.

Table 4. Experiment with $R_{\mathbf{fam}}$. All examples were used.

	min sup	20%	40%	60%	80%
MAPIX [5]	#patterns	55	31	23	11
	time (sec)	0.04	0.01	0.01	0.01
algo. in [4]	#patterns	441	153	51	15
	time (sec)	6.23	0.45	0.06	0.03
our algo.	#patterns	4601	1063	109	17
	time (sec)	9.55	0.54	0.08	0.01

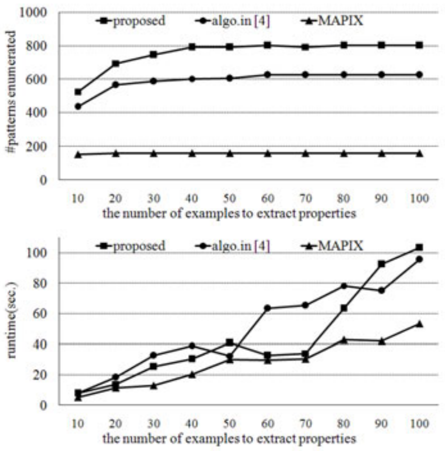


Fig. 2. The number of patterns and runtime as the number of sampled examples in Bongard.

References

1. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules. In: VLDB, pp. 487–499 (1994)
2. Dehaspe, L., De Raedt, L.: Mining association rules with multiple relations. In: Džeroski, S., Lavrač, N. (eds.) ILP 1997. LNCS, vol. 1297, pp. 125–132. Springer, Heidelberg (1997)

3. Dehaspe, L., Toivonen, H.: Discovery of frequent Datalog patterns. Data Mining and Knowledge Discovery 3(1), 7–36 (1999)
4. Inuzuka, N., Motoyama, J., Urazawa, S., Nakano, T.: Relational pattern mining based on equivalent classes of properties extracted from samples. In: Washio, T., Suzuki, E., Ting, K.M., Inokuchi, A. (eds.) PAKDD 2008. LNCS (LNAI), vol. 5012, pp. 582–591. Springer, Heidelberg (2008)
5. Motoyama, J., Urazawa, S., Nakano, T., Inuzuka, N.: A mining algorithm using property items extracted from sampled examples. In: Muggleton, S.H., Otero, R., Tamaddoni-Nezhad, A. (eds.) ILP 2006. LNCS (LNAI), vol. 4455, pp. 335–350. Springer, Heidelberg (2007)

Learning from Noisy Data Using a Non-covering ILP Algorithm

Andrej Oblak and Ivan Bratko

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

Abstract. In this paper we describe the non-covering inductive logic programming program HYPER/N, concentrating mainly on noise handling as well as some other mechanisms that improve learning. We perform some experiments with HYPER/N on synthetic weather data with artificially added noise, and on real weather data to learn to predict the movement of rain from radar rain images and synoptic data.

1 Introduction

In this paper we describe the non-covering ILP program HYPER/N, which is based on HYPER [1], but with added mechanisms for handling noisy data as well as some mechanisms that improve the computational efficiency. A non-covering ILP algorithm works with hypotheses as a whole, whereas covering ILP algorithms such as FOIL [6], Progol [5], and Aleph [8] build hypotheses by iteratively adding clauses to the hypothesis using greedy search. Advantages of a non-covering approach are the better ability to learn recursive hypotheses and the ability to learn multiple predicates simultaneously. The main disadvantage is a high combinatorial complexity. Besides that, HYPER's main disadvantage is lack of mechanisms to enable the learning from noisy data. We addressed this deficiency in HYPER/N.

The rest of the paper is organized as follows. In Section 2 we provide some related work. In Section 3 HYPER/N's mechanisms and improvements in comparison with HYPER are described. In Section 4 we perform experiments with HYPER/N and other systems in a weather prediction domain. Experiments were done on both synthetic data and real weather data. Synthetic data enabled controlled experiments with variable degree of noise in data. Section 5 summarizes the results.

2 Related Work

Not much work has been done in the field of learning from noisy data using non-covering ILP algorithms. In [4] a non-covering ILP system Lime is presented, which uses a Bayesian heuristic for inducing a hypothesis with the maximum posterior probability. In the experiments in [4] Lime, equipped with this heuristic, handled noise better than both FOIL and Progol. In [3] non-covering ILP system HYPER was used in a simple robotic domain, where an autonomous robot learns about the environment by performing experiments and collecting sensory data. The experiments were done with a real robot using an over-head camera to determine the position. So there was some noise in the measured coordinates. This noise was successfully handled by defining predicates in background knowledge in such a way that they tolerated some inaccuracy in numerical data.

3 HYPER/N

HYPER [1] is a non-covering ILP program. It starts with some overly general hypotheses that are complete also inconsistent, and it builds a refinement graph from them using three basic types of refinements of hypotheses:

– by matching two variables of the same type in a clause, e.g. $X1 = X2$,
– by refining a variable in a clause into a background term (list, constant, ...),
– by adding a background literal to a clause.

Since each refinement is a *specialization*, a successor of a hypothesis in the refinement graph only covers a subset of cases covered by the hypothesis' predecessor, it suffices that, during search, we only consider complete hypotheses. An incomplete hypothesis can never be refined into a complete one [1].

The main advantages of HYPER in comparison to other ILP programs is the ability to learn recursive and multi-predicate hypotheses well. The main disadvantage of HYPER are the high combinatorial complexity and inability to learn from noisy data. To a certain degree we have solved this in the improved version of HYPER, called HYPER/N.

Before we continue let us define what kind of noise we want HYPER/N to handle. Noise is considered as incorrectly classified examples, a positive example that is classified as a negative example and vice versa.

3.1 Handling of Noisy Data in HYPER/N

The first thing we had to do, for HYPER/N to handle noisy data, was to relax the conditions for completeness and consistency. Let us assume that in our learning set there are N_{noise} noisy examples. We say that a hypothesis is *approximately complete* if it covers all but N_{noise} positive examples, and *approximately consistent* if it covers N_{noise} negative examples. Before the start of learning the user must estimate the number of noisy examples in absolute terms (e.g. 13 noisy examples) or relative terms (e.g. 8% of learning examples are noisy). Too low estimations often result in no learned hypotheses, and too high estimations result in many learned hypotheses, since there are several possible coverings of a positive learning example set with such approximately complete hypotheses. It is up to the user to experiment with the estimations of noise to find an appropriate value.

As mentioned in the previous paragraph, HYPER/N can learn several different hypotheses. HYPER/N stores all the approximately complete and approximately consistent hypotheses found in a *list of potentially good hypotheses* together with the sets of positive examples, covered by those hypotheses. Let us assume that at one step of learning we have a list of potentially good hypotheses $H = \{H_0, H_1, ..., H_i\}$ and a list $E = \{E_{\mathrm{H}_0}, E_{\mathrm{H}_1}, ..., E_{\mathrm{H}_i}\}$ of sets of positive examples, covered by those hypotheses. For any new approximately complete and approximately consistent hypothesis H_j, $H_j \notin H$, with a set of covered positive examples E_{H_j}, we check: $E_{\mathrm{H}_j} \nsubseteq E_{\mathrm{H}_i}$ for all $E_{\mathrm{H}_i} \in E$. If true, hypothesis H_j is added to the list H and set E_{H_j} is added to the list E. With this we avoid storing hypotheses that cover same sets or subsets of positive examples as some other hypothesis, that is already stored in the list.

In HYPER learning simply stops when a complete and consistent hypothesis is found. But when learning from noisy data with HYPER/N, there is a high possibility that a complete and consistent hypothesis will not be found. Therefore before the learning starts, the user must set a maximum number of refinements. If a complete and consistent hypothesis is found before the number of refinements reaches this limit, such hypothesis is returned as a result. Otherwise an entire list of potentially good hypotheses is returned when HYPER/N reaches the maximum number of refinements. After that it is up to the user to decide what to do with the result – use it as a final solution or try different parameter values.

3.2 Learning Multiple Clauses from Noisy Data

Described modifications work fine if we are learning hypotheses consisting of only one clause. But if we want to learn a hypothesis consisting of two or more clauses, a certain problem arises. At each refinement HYPER first looks for a clause that alone covers at least one negative example [1]. But if we have a situation as illustrated in Fig. 1, where the first clause covers one noisy negative example, it may happen that learning will get stuck at that point.

Let us assume that no possible refinement exists that would exclude the noisy negative example from being covered by the first clause. HYPER always selects the first clause that covers a negative example for refinement. In the case of noise-free data this is sensible, but it is obvious that in the case of noisy data,

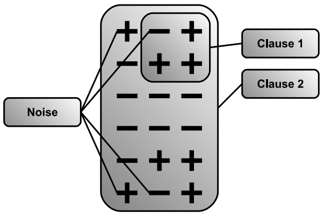


Fig. 1. Illustration of a problem that arises when learning hypotheses with more than one clause. HYPER always chooses the first clause for refinement that alone covers a negative example. This may cause the learning to get stuck that point.

the second clause is much more promising for refinement than the first one. HYPER/N avoids such cases by refining all clauses with highest cost, calculated by (1), where n is the number of negative, and p the number of positive examples covered by the clause alone.

$$clause\_cost = \frac{n}{p+n} \quad \text{if } n+p>0 \text{ otherwise } clause\_cost = 0 \ . \tag{1}$$

Breaking down the hypotheses into separate clauses and evaluating them separately is against HYPER's philosophy because HYPER works with hypotheses as a whole. It can also give us some dubious results when evaluating clauses from recursive hypotheses, since a general clause without a base clause usually covers no examples (positive and negative). Because of that we also tested another approach, similar to the 'gain' heuristic [2]. We evaluated hypothesis H with and without clause C. We counted that clause C covers all examples that were covered with hypothesis H and are not covered anymore with hypothesis $H \setminus \{C\}$. By counting this way we obtained p and n, and calculated clause cost using (1). While this approach may be more in line with HYPER's philosophy, HYPER/N had much more trouble learning recursive hypotheses (longer search or even no hypotheses learned at all) than by breaking down hypotheses into separate clauses and then evaluating them.

3.3 Some Additional Improvements

The next issue was to alleviate HYPER's shortsightedness. For this reason we implemented a lookahead with user-specified depth. When refining a hypothesis, HYPER/N refines also all hypothesis' refinements, ignoring heuristic estimates and repeats this for a specified number of levels. After that it removes all duplicates, adds all refined hypotheses to the list of "open" hypotheses and repeats this cycle with the next best hypothesis (one with lowest heuristic estimate).

A final improvement addresses the issue of high complexity. In certain cases, prior to learning the user knows certain constraints regarding the target theory. For example when learning in the weather domain we cannot have two different measurements of air pressure from one weather station at the same time. HYPER/N enables the user to provide such constraints as Prolog programs, which further increases HYPER/N's flexibility while at the same time decreases complexity, since all such hypotheses that do not satisfy the constraints are automatically discarded.

4 Experiments with Weather Data

We tested how HYPER/N performs at learning in a weather domain. In this domain we have a series of radar images (part of which is displayed in Fig.2), taken in 10 minute intervals, indicating the intensity of rain (none, light, medium, heavy and extreme) as well as a database with the synoptic measurements (air temperature, air pressure, wind direction, ...). Synoptic measurements were taken in 1 hour intervals at best. Spacial data was represented on a rectangular grid of

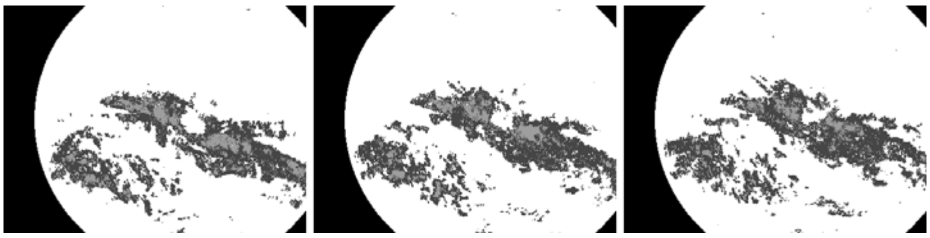


Fig. 2. Excerpt from a sequence of radar images, used for learning

cells. Besides this data we provided HYPER/N also with neighbor relations between cells and with successor in time relation. Many relations between synoptic measurements also exist and we have some prior knowledge about this domain. Therefore ILP seems appropriate for this domain since as ILP's strengths lie in relational learning and providing background knowledge in a very natural way.

The first thing we did was to verify how HYPER/N handles noise in the weather domain. The easiest way to do this was with synthetic data where we could control the amount of noise. We generated 200 learning examples using the following theory:

```
rain(any_type,[X,Y],T1) :-
  succ(T,T1), rain(any_type,[X1,Y1],T),
  neighbor(north,[X1,Y1],[X,Y]), wind_dir(north,T).

rain(light,[X,Y],T1) :-
  succ(T,T1), rain(any_type,[X1,Y1],T),
  neighbor(east,[X1,Y1],[X,Y]), wind_dir(east,T).
```

where `rain(any_type,[X,Y],T)` indicates that rain of any type (light, medium, heavy or extreme) is falling in the cell with coordinates `[X,Y]` at time `T`, `succ(T,T1)` indicates that `T1` is `T`'s successor in time, `neighbor(north,[X1,Y1],[X,Y])` indicates that the cell with the coordinates `[X,Y]` is the north neighbor of the cell with the coordinates `[X1,Y1]` (similar holds for `neighbor(east,[X1,Y1],[X,Y])`) and `wind_dir(north,T)` indicates that in the time `T` the wind is blowing towards the north (similar holds for `wind_dir(east,T)`). This theory says that rain in a cell will move to its neighbor cell in the wind direction.

To simulate noise, we changed the direction of rain movement in n examples. When the expected amount of noise was set to n, HYPER/N always learned the original theory. Under these conditions, HYPER/N managed to learn the correct hypothesis for the values of n up to approximately 50% of the number of positive learning examples. Setting the expected noise level lower than n resulted in no hypotheses learned when HYPER/N reached the number of maximum refinements (usually 1000 refinements). Setting it higher (approximately $n +$ 5 and more) resulted in several different hypotheses learned. Neither of those hypotheses was good as a whole.

The next step was to run HYPER/N on actual radar images. Since those images have resolution of 300 by 400 pixels, where each pixel represents 1km^2 in nature, we had for complexity reasons to reduce the amount of data. We discretized the images in smaller, 5 by 5 pixels cells, to decrease the number of learning examples (from 120.000 to 4.800 – for one radar image). Experimentally we found that in the weather domain HYPER/N could handle up to a few thousand learning examples. Increasing this number considerably resulted in very slow learning. Increasing the cell size was not an option since rain movement was not visible in too large cells, so we focused only on a smaller region around a weather station that produced synoptic measurements most regularly (in 1 hour intervals). So we selected a sequence of images (3.400 learning examples), where the rain mass is first moving towards the north, and after 3 hours started moving towards the east. HYPER/N induced the following hypotheses:

```
rain( rain_predicted, [X,Y],T1) :-
  succ( T, T1),  rain( rain_observed, [X1,Y1],T),
  wind_dir( D, T),   neighbor( D, [X1,Y1], [X,Y]).
```

and

```
rain(rain_predicted,[X,Y],T1) :-
  succ(T,T1),  rain(rain_observed,[X,Y],T).
```

The first hypothesis says that the rain moves in the direction in which the wind blows. The second hypothesis says that the rain "stays where it was". The learning times were in the order of an hour when the maximum number of clauses was "wastefully" set to 2 (1 would suffice).

We evaluated these hypotheses on 7.400 test data (74 radar images where each contains 100 cells in the observed area) where rain moves in several different directions. The first hypothesis (the one that says that the rain moves in the direction in which the wind blows) had a classification accuracy of 83%. The second one (the one that says that the rain stays where it was) had a classification accuracy of 89,8%. For comparison – a hypothesis that always predicts rain has a clasification accuracy of 46,3% and hypothesis that always predicts no rain has clasification accuracy of 53,7%.

For comparison with other ILP systems, we experimented with Aleph [8], Progol [5] and Alchemy [7] on the same reduced set of real weather data. We here present the most successful trials, obtained with Aleph. We ran Aleph with both the "classical" covering algorithm, and a non-covering approach with learning first-order decision trees. We did not get any success with the other non-covering algorithm in Aleph, called "theory induction", which induces a whole theory at once (rather than clause by clause) which would be most relevant for comparison with HYPER.

In the covering algorithm, an appropriate setting of parameter "noise" was 400, which resulted in a theory whose first clause is:

```
rain( YesNo, Place, Time) :-
   succ_start( Time0, Time),  rain( YesNo, Place, Time0).
```

The first clause is equivalent to HYPER/N's second theory. Then followed a large set of empty-body clauses that covered the false positive examples by simple enumeration. Aleph's theory has test set accuracy 0.888, very similar to HYPER/N's 0.898. The small difference is due to the clauses that cover single false positive examples. A user might sensibly remove these spurious clauses, leading to the same result as HYPER/N second theory. We did not succeed in making Aleph with covering induce HYPER/N's first theory as alternative.

Inducing class probability trees with parameter lookahead set to 1 just resulted in the unsatisfactory root-only tree. Setting lookahead to 2 or 3 gives tree structured rules:

```
rain( YesNo, Place, Time) :-
   succ_start( Time0, Time), rain( YesNo, Place, Time0),
   random(YesNo, [0.534425-no, 0.465575-yes]).

rain( YesNo, Place, Time) :-
   not (succ_start( Time0, Time), rain( YesNo, Place, Time0) ),
   random(YesNo, [0.507671-no, 0.492329-yes]).
```

The rules are sensible again (similar to theory 2 by HYPER). The class probability distribution in the first rule is confusing - it would best be omitted because when the body of the rule is satisfied, YesNo is already instantiated. Setting lookahead = 4 results in a theory that contains the same idea as theory 1 by HYPER/N. In conclusion, Aleph with appropriate settings and used in a "constructive" way induced theories similar to those by HYPER/N. Aleph was much faster than HYPER/N, with learning times in the order of a minute.

We were not able to obtain any success with Alchemy. A typical result is the following theory (after notational modification) that is hard to understand or use:

```
   not neighbor( Dir, [X1,X2], [X1,X1]) v
   not rain( Type, [X1,Y], Time) v not rain(Type, [X2,Y], Time)
```

It should be admited that we had problems using these systems, specially with searching for good parameter setting. Better settings may exist than those we used.

5 Conclusion

In the paper we presented the mechanisms in HYPER/N that enable learning from noisy data and on larger domains than HYPER. We studied HYPER/N performance by experimnets in a real weather domain. In comparison with other popular ILP programs such as Aleph [8], FOIL [6] and Progol [5] as well as Alchemy [7], HYPER/N's advantage is its better ability to learn recursive clauses and to learn multiple predicates simultaneously. Experiments in the weather domain show how important it is to correctly assess the degree of noise in the data.

The accuracy and appropriate structure of induced theories largely depend on this estimate. Future work will surely include automation of some mechanisms, such as finding an appropriate value for the expected amount of noise, to make it more robust and user friendly.

Acknowledgment

This work was partly done in the X-Media project (www.x-media-project.org) funded by the European Commission, EC grant IST-FP6-026978, and Slovene Agency for Research and Development (ARRS). The Slovene environmental agency ARSO provided the weather data used in this study. We thank our colleagues Jure Žabkar, Martin Možina, Marko Žerjal and Matic Standeker for discussion and other help in this work.

References

1. Bratko, I.: Inductive Logic Programming, 3rd edn. Prolog Programming for Artificial Intelligence, ch. 19, pp. 484–519. Addison Wesley, Reading (2001)
2. Lavrač, N., Džeroski, S., Bratko, I.: Handling Imperfect Data in Inductive Logic Programming. In: De Raedt, L. (ed.) Advances in Inductive Logic Programming, pp. 48–64. IOS Press, Amsterdam (1996)
3. Leban, G., Žabkar, J., Bratko, I.: An Experiment in Robot Discovery With ILP. LNCS, pp. 77–90. Springer, Heidelberg (2008)
4. McCreath, E., Sharma, A.: ILP with Noise and Fixed Example Size: A Bayesian Approach. In: Fifteenth International Joint Conference on Artificial Intelligence, pp. 1310–1315. Morgan Kaufmann Publishers, San Francisco (1997)
5. Muggleton, S.: Inverse entailment and Progol. New Generation Computing 13, 245–286 (1995)
6. 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)
7. Richardson, M., Domingos, P.: Markov Logic Networks. Machine Learning 62, 107–136
8. Srinivasan, A.: The Aleph Manual. Computing Laboratory, Oxford University (2007)

Can HOLL Outperform FOLL?

Niels Pahlavi and Stephen Muggleton

Department of Computing, Imperial College London,
180 Queen's Gate, London SW7 2BZ, UK
{niels.pahlavi,s.muggleton}@imperial.ac.uk

Abstract. Learning first-order recursive theories remains a difficult learning task in a normal Inductive Logic Programming (ILP) setting, although numerous approaches have addressed it; using Higher-order Logic (HOL) avoids having to learn recursive clauses for such a task. It is one of the areas where Higher-order Logic Learning (HOLL), which uses the power of expressivity of HOL, can be expected to improve the learnability of a problem compared to First-order Logic Learning (FOLL). We present a first working implementation of λProgol, a HOLL system adapting the ILP system Progol and the HOL formalism λProlog, which was introduced in a poster last year [15]. We demonstrate that λProgol outperforms standard Progol when learning first-order recursive theories, by improving significantly the predictive accuracy of several worked examples, especially when the learning examples are small with respect to the size of the data.

1 Introduction and Motivations

[3] describes Higher-order Logic (HOL) as "a natural extension of first-order logic (FOL) which is simple, elegant, highly expressive, and practical" recommends its use as an "attractive alternative to first-order logic". HOL allows for quantification over predicates and functions and is intrinsically more expressive than FOL. Much of logic-based Machine Learning research is based on FOL and Prolog, including Inductive Logic Programming (ILP). Yet, HOL, and has been seldom used. According to [9], "the logic programming community needs to make greater use of the power of higher-order features and the related type systems. Furthermore, HOL has generally been under-exploited as a knowledge representation language". In [9], the use of HOL in Computational Logic, which has been "advocated for at least the last 30 years" is illustrated: functional languages, like Haskell98; Higher-order programming introduced with λProlog [11]; integrated functional logic programming languages like Curry or Escher; or the higher-order logic interactive theorem proving environment "HOL". It is also used in IBAL and for Deep Transfer Learning.

We have decided to adapt ILP within a HOL framework, developing Higher-order Logic Learning (HOLL). ILP seems to be rather intuitively adaptable to a FOL formalism. We present a first working implementation of λProgol, a HOLL system adapting the ILP system Progol and the HOL formalism λProlog. We decided to choose Higher-order Horn Clauses (HOHC) [13] as a HOL formalism, since it is one of the logical foundations of λProlog. As a ILP system, we chose to adapt Progol [12], which is a popular and efficient implementation. We want to determine whether HOLL can outperform FOLL and study the trade-off that there may be between learnability and searching costs (the use of Henkin semantics as in [17], seems to alleviate these and maintain the structure of the search space).

There have been attempts to use HOL for logic-based Machine Learning such as by Harao starting in [6], Feng and Muggleton [4] and Furukawa and Goebel [5]. They provide different higher-order extensions of least general generalization in order to handle higher-order terms in a normal ILP setting, whereas we use λProlog, a HOL framework, as a logical foundation to extend first-order ILP to a higher-order context. The main similar work is [9] by Lloyd and Ng, where higher-order machine learning is also developed. It uses a typed higher-order logic, but, although similar, "a different sublogic is used for λProlog programs than the equational theories proposed" in [9]. It details a learning system, called *ALKEMY*. A main difference is that Lloyd's approach is not based on Logic Programming and therefore on ILP. According to Flach, "it is almost a rational reconstruction of what ILP could have been, had it used Escher-style HOL rather than Prolog"; whereas we intend, through the use of higher-order Horn clauses to keep the Horn clauses foundations of LP and ILP and to extend it.

HOLL will be tested and assessed on new problems and applications not learnable by ILP (which includes the learning of higher-order predicates), but also on how well it performs on problems already handled by ILP to compare it with existing ILP systems in order to determine if it can outperform FOLL. It would be therefore of interest to look at learning problems not handled well by ILP. One of these is learning tasks involving recursion. According to [10], "learning first-order recursive theories is a difficult learning task" in a normal ILP setting. However, we can expect a higher-order system to learn better than a first-order system on such problems, because we could use higher-order predicates as background knowledge to learn recursive theories (a similar approach is already used for some recursive functions in the Haskell prelude); and it will be sounder, more natural and intuitive, hence probably more efficient than meta-logical features which come from functional languages. We can for example use the higher-order predicate *trans* as background knowledge to learn naturally the recursive predicate *ancestor* (see Sect. 3). More generally, the expressivity of HOL would make it possible to represent mathematical properties like symmetry, reflexivity or transitivity, which would allow to handle equational reasoning and functions within a logic-based framework. We could also represent such properties in the following fashion (in the case of symmetry) : R@X@Y <= [sym@R,R@Y@X], and, abduce for example that the move of the bishop in chess is symmetric: sym@bishop_move.

About the use of probability in a logic-based setting, [14] advocates for probability to be captured directly in the theory itself, which can be done naturally and directly with HOL, as opposed to almost all approaches having a clear separation between the logical statements and the probabilities.

2 λProgol: A Higher-Order ILP System

In this section, λProgol, a higher-order ILP formalism is presented. It is based upon Progol and Mode-Directed Inverse Entailment as defined in [12]. However, it generalizes this approach on HOHC and λProlog. λProlog, developed by Miller and Nadathur, is a higher-order logic programming language handling polymorphic typing, scoping over names and procedures, modular programming, abstract data types, the use of lambda terms as data structures and, more importantly for this paper, higher-order programming. λProlog is based on HOHC, introduced in [13] and defined as "a generalization of Horn clauses to a higher-order logic" and a "basis for logic programming". According to [13], HOHC can be "characterized as those obtained from first-order goal formulas and definite sentences by supplanting first-order terms with the terms of a typed λ-calculus and by permitting quantification over function and predicate symbols".

Since our implementation is in Prolog, a λProlog interpreter in Prolog is needed. It is based on a theorem proving procedure for HOHC outlined in [13], based on Huet's unification algorithm in typed λ-calculus [8], and whose soundness is proved. The main differences in the λProgol algorithm, compared to Progol, come from this interpreter and from the fact that it requires background knowledge and examples to be not Horn clauses but λProlog clauses.

Definition 1. ***λProlog Interpreter.***
A λProlog clause is of the form $(HeadAtom \Leftarrow [BodyAtom_1, \ldots, BodyAtom_n])$, where HeadAtom has to be rigid. A λProlog formula is one of the following: (1) A variable or a constant, (2) (X/F), where F is a formula, (3) (F1@F2) where F1 and F2 are formulae, (4) (sigma F), where F is a formula, (5) (pi F), where F is a formula. sigma and pi represent respectively the existential and universal quantifiers. / represents abstraction and @ represents function application as it is defined in λ-calculus [1] and in λProlog. Atomic formulas must have the form $(h@t_1@\ldots@t_l)$, where h is either a variable or constant and $t_1 \ldots t_l$ are terms. If h is a constant, it is a rigid atom; if h is a variable, it is a flexible atom. A list is of the form $cons@el_1@\ldots@el_m@nil$. nil is the empty list.

The λProgol algorithm is constituted by three algorithms (Algs. 1, 2 and 3) which are very similar to the Progol algorithms (only Alg. 1 will be detailed in this paper). The mode declarations and mode language are identical to Progol (Definitions 20, 21, 22 in [12]) except that the mode atoms can be different because λProlog atoms are different from FOL atoms. The construction of $\bot_i$, which is the least general element of the bounded sub-lattice for each example e is described in Alg. 1. i represents the maximum variable depth determining how many times step 5 is executed; *Recall* determines how many times the λProlog interpreter is called for each instantiation of the clause in step 4. The line 5.a.i in the algorithm is specific to λProgol, it is to prevent the call of flexible atoms by the λProlog interpreter. Indeed, the type *pred* is set to correspond to higher-order predicate, which can be uninstantiated (i.e. still variable) when called by the λProlog interpreter. The call to $pred(u)$ instantiates these variables.

Algorithm 1. Construction of $\bot_i$.

```
1. Given natural numbers $i$, λProlog clauses $B$, λProlog clause $e$ and set of mode
   declarations $M$.
2. Let $k = 0$, $hash : Terms \to N$ be a hash function which uniquely maps terms to
   natural numbers, $\overline{e}$ be $\overline{a} \wedge b_1 \wedge \ldots \wedge b_n$, $\bot_i = \langle\rangle$ and $InTerms = \emptyset$.
3. If there is no modeh in $M$ such that $a(m) \preceq a$ then return the empty clause $\square$.
   Otherwise let $m$ be the first modeh declaration in $M$ such that $m$ subsumes $a$ with
   substitution $\theta_h$. For each $v/t$ in $\theta_h$
   (a) if $v$ corresponds to a $\#type$ then replace $v$ in $m$ by $t$
   (b) otherwise replace $v$ in $m$ by $v_k$ where $k = hash(t)$ and
   (c) add $t$ to $InTerms$ if $v$ corresponds to $+type$.
   Add $m$ to $\bot_i$.
4. If $k = i$ return $\bot_i$ else $k = k + 1$.
5. For each modeb $m$ in $M$, let $\{v_1, \ldots, v_n\}$ be the variables of $+type$ in $m$ and
   $T(m) = T_1 \times \ldots \times T_n$ be a set of $n$-tuples of terms such that each $T_i$ corresponds to
   the set of all terms from $InTerms$ of the type associated with $v_i$ in $m$ ($t$ is tested
   to be of a particular type by calling $type(t)$ with the λProlog interpreter).
   (a) For each $\langle t_1, \ldots, t_n\rangle$ in $T(m)$ and $\theta = \{v_1/t_1, \ldots, v_n/t_n\}$. Repeat recall times:
       i. for every variable $u$ in $m\theta$ of type $pred$, add the call $pred(u)$ to the λProlog
          interpreter
       ii. if the λProlog interpreter succeeds on goal $m\theta$ with answer substitution
           $\theta'$then for each $v/t$ in $\theta$ and $\theta'$ if $v$ corresponds to a $\#type$ then replace $v$
           in $m$ by $t$ otherwise replace $v$ in $m$ by $v_k$ where $k = hash(t)$ and add $t$ to
           $InTerms$ if $v$ corresponds to $-type$. Add $\overline{m}$ to $\bot_i$.
6. Goto step 4.
```

best(s), *prunes*(s), *terminated*(s) are defined like in Progol to find a clause with maximal compression. $\rho(s)$ is currently defined like in Aleph. Alg. 2 corresponds to the search for a single clause in the subsumption lattice and Alg. 3 is a simple cover set algorithm as in, respectively, Algs. 42 and 44 in [12], except that the unflattening is not done. Currently, only the *cover* searching has been implemented.

An implementation of λProgol has been made, has been tested, and is available at [16]. Our first choice of implementation was based on λProlog but revealed to be too inconvenient and inefficient to use; instead the current implementation is in Prolog, which is more convenient and more efficient; a requirement is the use of a λProlog interpreter, which was implemented using a depth-first approach.

3 Results and Applications

In this section, we present the initial results that we have obtained so far about learning recursive theories. Learning first-order recursive theories remains a difficult learning task in a normal Inductive Logic Programming (ILP) setting, although numerous approaches have addressed it. These approaches are listed in [10] and include FOIL and Progol. MPL and ATRE [10] are two multiple predicate learning systems that address this problem. In [7], IGOR II, an analytical inductive functional system "specialised to learn recursive programs" is described and is compared to GOLEM and the search-based inductive functional system MagicHaskeller.

Since we are interested in determining if learning within a higher-order context can improve the learning of a given problem, we have decided to compare λProgol with the first-order system it is based upon, i.e. Progol. Both systems having almost the same learning algorithms, the comparison will stress the difference between FOLL and HOLL on a given learning task.

To ensure the fairness of the comparison and the soundness of the learning, we compare Progol and λProgol in standard settings. We will therefore neither include the presence of meta-logical features for Progol, nor the use of clauses of the type holds(...) that make use of first-order variables in order to simulate higher-order variables. In order for Progol to learn recursive clauses, we will use *implicational* searching instead of *cover* searching.

The learning task (used in [10]) we are going to develop consists of learning the predicate *ancestor* given a genealogical tree defined by facts for the predicates *parent* and *married* as background knowledge and positive and negative examples of the predicate *ancestor*. It is illustrative of how the power of expressivity of HOL could be used to improve the learnability of a problem.

The genealogical tree used for this experiment is described in Fig.1 and in [2]. It contains 119 members over 11 relations of the Romanov imperial Russian dynasty. The Progol input file (except the *ancestor* examples) is described in Ex. 1. ((...) corresponds to parts that have omitted).

Example 1. **Progol input file for learning ancestor.**

Mode declarations:
```
:- modeh(*,ancestor(+person,+person))?
:- modeb(*,parent(+person,+person))?
:- modeb(*,parent(+person,-person))?
:- modeb(*,ancestor(+person,-person))?
```
Type declarations:
```
person(X) :- male(X). person(X) :- female(X).
female(eudoxia_streshneva). female(maria_dolgorukova). (...) female(anastasia_1).
male(michael_I). male(alexis_I). (...) male(michael_3). male(alexei_3).
```
Background Knowledge:
```
parent(michael_I,alexis_I). (...) parent(alexandra_fyodorovna,alexei_3).
married(michael_I,eudoxia_streshneva). (...) married(michael_3,natalya_wulffert).
```

In Ex. 1, we can note that modeb(*,ancestor(+person,-person)) suggests a recursive definition for *ancestor*, and that the presence of only *parent* in the modebs declarations will force Progol to learn *ancestor* with *parent* and not *married.* The λProgol input file (except the *ancestor* examples) is described in Ex. 2.

Example 2. **λProgol input file for learning ancestor.**

Mode declarations:
```
:- modeh(*,ancestor@(+person)@(+person)).
:-modeb(*,(#pred_secondorder)@(#pred_person_person)@(+person)@(+person)).
```
Type declarations:
```
pred_secondorder(trans).
```

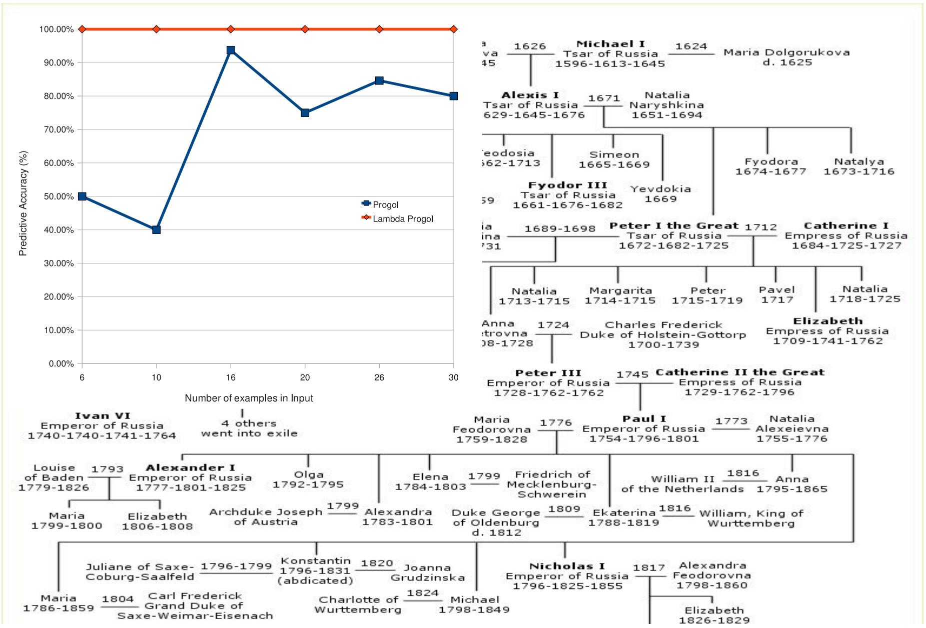


Fig. 1. Left: Comparison between Progol and Lambda Progol on the Ancestor example. Right: Part (around one third) of the Romanov dynasty tree used in the experiments.

pred_person_person(parent). pred_person_person(married).
person(X) :- male(X). person(X) :- female(X).
female(eudoxia_streshneva). female(maria_dolgorukova). (...) female(anastasia_1).
male(michael_I). male(alexis_I). (...) male(michael_3). male(alexei_3).
Background Knowledge:
trans@R@X@Y <= [R@X@Y]. trans@R@X@Z <= [R@X@Y,trans@R@Y@Z].
parent@michael_I@alexis_I <= []. (...) parent@alexandra_fyodorovna@alexei_3
married@michael_I@eudoxia_streshneva <= []. (...) married@michael_3@ natalya_wulffert

In Ex. 2, we can observe that the modeh is similar to the one in Progol. There is only one modeb however as the learning is not recursive, the modeb asks for a constant second order predicate that has for arguments a constant (first-order) predicate over two persons, and two input persons. The use of the predicate *parent* is not mandatory and the predicate *married* will be tried, contrary to what is done in Progol. Compared to the Progol input, the types of the second-order predicate *trans* and the predicates *parent* and *married*, the other types are identical. As for the background knowledge, the definition of *trans* is added and the rest corresponds to the same *parent* and *married* facts in λProgol notations.

To compare the two systems, we created files containing positive and negative examples of *ancestor*. These files contain an equal number of positive and negative examples generated randomly and they contain respectively 6 (described in Ex. 3.), 10, 16, 20, 26 and 30 examples in total.

Example 3. **Progol and λProgol example files for learning ancestor.**
Progol example file:
:-ancestor(natalia_naryshkina,alexis_I). :-ancestor(joseph_austria,paul_I).
:-ancestor(charlesleopold_mecklenburg,marie_mecklenburg).
ancestor(maria_feodorovna,alexei_2). ancestor(catherine_II,tatiana_1).
ancestor(eudoxia_streshneva,konstantin_1).
λProgol example file:
false <= [ancestor@natalia_naryshkina@alexis_I]. false <= [ancestor@joseph_austria@paul_I].
false <= [ancestor@charlesleopold_mecklenburg@marie_mecklenburg].
ancestor@maria_feodorovna@alexei_2 <= []. ancestor@catherine_II@tatiana_1 <= []. ancestor@eudoxia_streshneva@konstantin_1 <= [].

We then compared the respective predicative accuracy of Progol and λProgol on these examples by doing a leave-one-out cross-validation. The results of this experiment is shown in Fig.1. All the input files and the outputs of the experiment can be found at [16]. In Progol, the definition of *ancestor* should be the two following clauses: ancestor(A,B) :- parent(A,B). and ancestor(A,B) :- parent(A,C), ancestor(C,B). However, during the cross-validation, Progol has rarely found this definition. It sometimes returns incorrect recursive definitions; but also non recursive definitions like ancestor(A,B) :- parent(A,C), parent(C,D), parent(D,E). The latter is due to the finiteness of the genealogical tree used. It often cannot induce clauses that compress the data. To learn the definition correctly, Progol needs to find the following positive examples: ancestor(X,Y) such that parent(X,Y) (for the base case) and both ancestor(A,C) and ancestor(B,C) such that parent(A,B) or parent(B,A) (for the recursive step). The larger the input and the smaller the number of examples, the smaller the probability to learn the definition correctly. Hence the difficulty to learn and the observation that the accuracy seems to decrease with the number of examples. On the other hand, λProgol learns the correct definition in all the cases, which is ancestor@X@Y ⇐ [trans@parent@X@Y] (from this definition, we can obtain, by unfolding and with the closed world assumption, the Progol definition). This definition is non recursive and can be learned from any given positive example.

On this example consisting of learning a recursive definition from large data with few examples, we have showed that HOLL can outperform FOLL. Moreover, the same predicate *trans*, as background knowledge, could be used to learn the predicate *less_than* given the predicate *successor*. *mappred*, which "given a predicate of two arguments and two lists, checks that corresponding elements of these two lists are related by the given predicate", and *foreach*, which "given a predicate of one argument and list, checks that every element of that list satisfies that predicate" can be used to learn predicates defined over lists. The high-order property of sortedness for an order R, which can be represented by the following: sorted@(cons@X@nil) <= [] and sorted@(cons@X@(cons@Y@Z)) <= [R@X@Y,sorted@Y@Z] can be used to learn words sorted lexicographically for example. Similarly, we can use the same higher-order predicate to both learn to reverse a list and to sum the elements of a list depending on the context. But a higher-order predicate can be used to learn an other higher-order predicate like *trans*. This is only a few of examples of HOLL which can outperform FOLL and these learning problems have been tested and can be found at [16].

4 Future Works

We intend to continue the tests and comparisons of λProgol against already existing ILP systems to determine how HOLL may outperform FOLL as it was shown above. We aim to present theoretical results for HOLL. ILP theory seems to be rather intuitively adaptable within a HOL framework. For λProgol, we will have to prove that higher-order inverse entailment is possible and to generalize correctness and complexity results for the Progol Bottom Clause and Search algorithms. In [17], a model-theoretic semantics for HOHC is provided. We also want to investigate tasks and discoveries not learnable by first-order ILP. It could be of interest to look at HOL theorem provers, or integrated functional logic programming languages and Mathematical Discovery. Further objectives may be to investigate abduction within λProgol, introduce Probability and adapt Probabilistic Logic Learning within HOL, look at applications such as Bioinformatics, where ILP has been successfully applied, and consider other logics within λProlog.

References

1. Barendregt, H.: The λcalculus its syntax and semantics. North-Holland, Amsterdam (1984)
2. Dynasty, R.: http://en.wikipedia.org/wiki/Tsars_of_Russia_family_tree
3. Farmer, W.: The seven virtues of simple type theory. J. Applied Logic (2008)
4. Feng, C., Muggleton, S.H.: Towards inductive generalisation in higher order logic. In: Proc. Ninth Int. Work. on Machine Learning, pp. 154–162 (1992)
5. Furukawa, K., Imai, M., Goebel, R.: Hyper least general generalization and its application to higher-order concept learning. Tech. report, Keio University (1996)
6. Harao, M.: Analogical reasoning based on higher-order unification. In: ALT (1990)
7. Hofmann, M., Kitzelmann, E., Schmid, U.: Analysis and evaluation of inductive programming systems in a higher-order framework. In: Dengel, A.R., Berns, K., Breuel, T.M., Bomarius, F., Roth-Berghofer, T.R. (eds.) KI 2008. LNCS (LNAI), vol. 5243, pp. 78–86. Springer, Heidelberg (2008)
8. Huet, G.: A unification algorithm for typed λcalculus. Theor. Comp. Sci. (1975)
9. Lloyd, J.W.: Logic for Learning. Springer, Berlin (2003)
10. Malerba, D.: Learning recursive theories in the normal ILP setting. Fundam. Inform. 57(1), 39–77 (2003)
11. Miller, D.: λProlog: An Introduction to the Language and its Logic (1998)
12. Muggleton, S.H.: Inverse entailment and Progol. New Generation Computing 13, 245–286 (1995)
13. Nadathur, G., Miller, D.: Higher-order Horn Clauses. Journal of the ACM (1990)
14. Ng, K.S., Lloyd, J.W., Uther, W.T.B.: Probabilistic modelling, inference and learning using logical theories. Ann. Math. Artif. Intell. 54(1-3), 159–205 (2008)
15. Pahlavi, N., Muggleton, S.: Higher-order Logic Learning. In: ILP 2009, Poster (2009)
16. Pahlavi, N.: λProgol Homepage, http://www.doc.ic.ac.uk/~namdp05/
17. Wolfram, D.A.: A semantics for λProlog. Theor. Comp. Sci., 277–289 (1994)

Incremental Learning of Relational Action Models in Noisy Environments

Christophe Rodrigues, Pierre Gérard, and Céline Rouveirol

LIPN/A^3, University of Paris 13
firstname.lastname@lipn.univ-paris13.fr

Abstract. In the Relational Reinforcement Learning framework, we propose an algorithm that learns an action model (or an approximation of the transition function) in order to predict the resulting state of an action in a given situation. This algorithm learns incrementally a set of first order rules in a noisy environment following a data-driven loop. Each time a new example is presented that contradicts the current action model, the model is revised (by generalization and/or specialization). As opposed to a previous version of our algorithm that operates in a noise-free context, we introduce here a number of indicators attached to each rule that allows to evaluate if the revision should take place immediately or should be delayed. We provide an empirical evaluation on usual RRL benchmarks.

Introduction

In this paper[1], we propose an algorithm that simultaneously tackles the problems of incrementality and indeterminism in action model learning when using relational representations for states and actions. A relational representation is expected to have better generalization capabilities, improved scaling-up and transfer of solutions since it does not rely on a number of attributes describing states nor on their order.

When a system is involved in a sensori-motor loop with its environment, it perceives its state, chooses and performs an action before perceiving its new situation, acting again and so on. Action models permit to anticipate the outcomes of any action in a given state. Such models are necessary for planning and may be used in Reinforcement Learning (RL) to speed up the overall learning of the optimal action [13].

This work takes place at the intersection of Relational Reinforcement Learning ([5], see [14] for a review) and Planning in first order logics [12]. In both fields, automatically acquiring action models from experience is a major concern.

The main originality of our work is that it tackles both incremental learning and learning in non deterministic environments. In [11], we already proposed incremental algorithms in the deterministic case, with a convergence proof. By incremental learning, we mean revising the model each time a new example contradicting the current action model is presented to the system, without storing every example and periodically re-running batch learning. Incremental learning is suitable for designing adaptive systems. In the deterministic case, the order of the provided examples might be misleading and yield inadequate generalization (or specialization) choices, in particular when

[1] This work was partially supported by the ANR project HARRY.

an action has disjunctive preconditions. The system therefore has to be provided with mechanisms for reconsidering early inadequate decisions.

The deterministic assumption is quite a strong one as, in real world applications, the outcomes of actions are often stochastic (noise, inadequate representations, partial observability treated as stochasticity, etc.). Furthermore, indeterminism and incrementality should be tackled together, considering that in a non deterministic context, the order in which examples are presented to the system is even more critical.

On the one hand, several works in the Relational RL framework [2] or in the planning field [7,1,12,17,15] propose to learn action models incrementally, but they all only consider deterministic environments. On the other hand, other works [10,9,18] address stochasticity, but are limited to batch learning. Based on KWIK [8], [16] addresses stochastic problems but can hardly be considered as incremental : all examples are stored and a new batch-learning is performed each time. Other related work do neither tackle incrementality nor indeterminism such as INTHELEX [6]. Figure 1 gives an overview of related work.

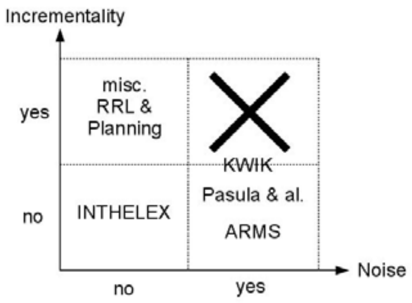


Fig. 1. Incrementality together with indeterminism

In Section 1, we detail our learning framework by describing the relational first order representation of the state and action spaces and introducing the general mechanisms of the algorithm presented in Section 2. Here, we provide an overview of the proposed incremental generalization and specialization mechanisms. Before concluding, the algorithms are empirically tested on the regular RRL benchmark environment.

1 Learning Problem

1.1 Action Model

An action model is a theory of the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ of a Markov Decision Process (MDP) with a set $\mathcal{S}$ of possible states and a set $\mathcal{A}$ of possible actions. Here, $\Pi(\mathcal{S})$ denotes a probability distribution over the state space. Unlike in the deterministic case, the outcomes of the same action in the same state might be different from time to time, due to noise or other concerns.

An example x for the learning process is composed of a given state, a chosen action and the resulting new state. We respectively note them as $x.s$, $x.a$ and $x.s'$. The *effect*

$x.e$ of an action describes what has changed in the state after applying the action, *i.e.* $x.e = \delta(x.s, x.s')$. Rather than strictly learning a model of the transition function, we model what changes when the action applies, by taking advantage of $x.s$, $x.a$ and $x.e$. Let us denote this model T.

Due to the model complexity, we have chosen not to learn the whole distribution probability over possible effects for a given action in a given state. We rather restrict the model T to most likely outcomes. We assume in this paper this will be sufficient to distinguish between noise and relevant information in most cases.

1.2 Relational Representation

Examples. Examples are described by a Datalog-like language (no function symbol but constants). Objects are denoted by constants (denoted in the following as a, b, $f \ldots$). Variables are denoted by upper-case letters (X, $Y \ldots$). In a noise-free context, when an agent emits an action, the effect part completely describes the effects of the action: literals not affected by the action remain unchanged. The effect, as usual in a STRIPS-like notation, is composed of two literal sets: $x.e.add$ is the set of literals getting true when the action is performed, and $x.e.del$ is the set of literals getting false when the action is applied. The examples can be noted $x.s/x.a/x.e.add$, $x.e.del$, with no negated atoms in $x.s$, $x.a$ and $x.e.add$, and $x.e.del$ described as a conjunction of negated literals.

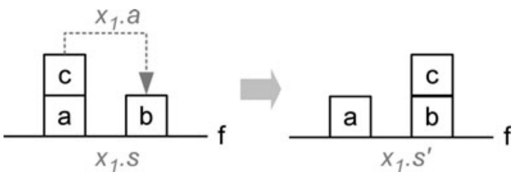


Fig. 2. In a simplified blocks world with only *on* and *move* predicates, states and action yielding to example x_1: $on(a, f), on(b, f), on(c, a)/move(c, b)/on(c, b), \neg on(c, a)$

Rules. Most works in RRL use instance based methods [3] or decision trees [4] to simultaneously represent an action model and the value function associated to this model. Existing instance based methods use predefined distances suited to the problems. Decision trees are top-down methods which – in the incremental case — highly rely on the order of presentation of the examples, thus leading to over-specializations.

In this paper, we propose a rule-based representation of the action model T , because inadequate generalization/specialization choices in an incremental context can be easily reconsidered when actions are represented by independent rules and that no ad-hoc prior knowledge is needed, as it is the case for instance based approaches.

Each rule r is composed of a precondition $r.p$, an action $r.a$ and an effect $r.e$. The precondition is represented by a conjunction of positive literals, which have to be satisfied so as to apply the rule. The action is a literal defining the performed action. As for examples, the effect is composed of two literal sets, $r.e.add$ and $r.e.del$. An action $r.a$ has no other effects but those described by $r.e$. In order to be *well formed*, a rule r must be such that i) $r.e.del \subseteq r.p$ ii) $r.e.add \cap r.p \neq \emptyset$ iii) $r.a$ and $r.e$ must be connected. Finally, all variables occurring in $r.a$ should also occur in $r.p$ and $r.e$, but $r.p$ and $r.e$ may refer to objects/variables not occurring in $r.a$. Rules can be denoted $r.p/r.a/r.e.add, r.e.del$. There is no negated literal in $r.p$, and each literal of $r.e.del$ is negated.

This *extended STRIPS* formalism we adopt in this work is more expressive than the regular STRIPS language, as considered for instance in [15]. For a given action and effect, it is possible to associate several preconditions, each expressed as a conjunction of literals. This is not the case in [15], which only accepts conjunctive preconditions for an action. Moreover, our formalism accepts action rules where variables/objects not occurring in the action literals may occur in preconditions and/or effects. Learning here includes learning the "schema" of STRIPS-actions (number of rules, exact variables involved in the action).

1.3 Matching and Covering

The following matching relationships all make use of OI-subsumption (subsumption under Object Identity) [6] as a *generality* relation. A formula G OI-subsumes a formula S iff there exists a substitution σ such that $G\sigma \subseteq S$, where σ is an injective substitution (two different variables of the domain of σ are assigned to different terms). For instance, $p(X, Y)$ does not OI-subsume $p(a, a)$ because X and Y cannot be assigned to the same constant.

The *pre-matching* relation $\overset{sa}{\sim}$ allows to decide whether a given rule may apply to predict the outcomes of a given example. For any rule r, state s and action a, $r \overset{sa}{\sim} (s, a)$ iff there exists injective substitutions σ and θ such that i) $r.a\sigma = a$ ii) $r.p\sigma\theta \subseteq s$.

The *post-matching* relation $\overset{ae}{\sim}$ permits to decide whether a given rule may explain a given state modification when a given action is performed. For any rule r, and action a and effect e, $r \overset{ae}{\sim} (a, e)$ iff there exists an inverse substitution ρ^{-1}, and two injective substitutions σ and θ such that i) $r.a\rho^{-1}\sigma = a$ ii) $r.e\rho^{-1}\sigma\theta = e$.

The above relations can be extended to matching between rules r and examples x, by adequately taking into account $x.s$, $x.a$ and $x.e$ instead of s, a and e in the definitions.

The *covering* relation $\approx$ permits to check whether an example can be accurately predicted by the model. For any rule r and example x, $r \approx x$ iff $r \overset{sa}{\sim} (x.s, x.a)$ and $r \overset{ae}{\sim} (x.a, x.e)$ for the same injective substitutions σ and θ.

An example x *contradicts* a rule r ($x \nsim r$) if r pre-matches $(x.s, x.a)$ for σ and θ substitutions, and r does not post-match $(x.a, x.e)$ with the same substitutions. In such a case, the rule incorrectly predicts the outcomes of the action.

2 Incremental Relational Learning of an Action Model

2.1 Sketch of the Algorithm

The system takes examples as defined in Section 1. The examples are presented incrementally, each one possibly yielding an update of the action model. The method we propose is example-driven and bottom-up. The starting point is thus an empty rule-set; the interactions between the system and its environment produce rules by computing least general generalization (*l*gg) under Object Identity between examples, and between rules and examples. This approach is different from — descendant — decision trees.

General rules are produced as the *l*gg of two rules/examples. When a *l*gg takes place, the resulting generalization keeps track of which rules/examples it comes from, each rule r therefore has a list of ancestors $r.anc$. Each ancestor rule might have ancestors as well, yielding a hierarchical structure.

This structure is used when specializing, in case an inadequate generalization is detected (see below). In that case, the corresponding rule is removed and replaced by one of its rule ancestors; specialization is called recursively until a trusted rule ancestor is reached.

Example ancestors that have been rejected during specialization are re-injected in the system for learning, and thus may lead to further revisions of the model. So as not to explore the same over-generalizations again, a tabu list is associated with such examples.

Let us suppose we observe a blocks world with three blocks and the floor f. By observing several moving actions, let us suppose the system has learnt the two following (correct) rules:

$$\begin{aligned} r_1 : &\, cl(X), cl(Y), on(X,Z), bl(X), bl(Y), bl(Z) \,/ \\ &\, move(X,Y) \,/\, on(X,Y), \neg cl(Y), \neg on(X,Z), cl(Z) \end{aligned}$$

for stacking the top X of a two blocks stack on a single block Y, provided that both X and Y are clear of blocks and

$$\begin{aligned} r_2 : &\, on(X,f), cl(X), cl(Y), on(Y,Z), bl(X), bl(Y) \,/ \\ &\, move(X,Y) \,/\, on(X,Y), \neg cl(Y), \neg on(X,f) \end{aligned}$$

for stacking a block X initially on the floor on another block Y (X and Y should also be clear of blocks). Let us now assume that the following noisy example occurs:

$$\begin{aligned} x_n : &\, on(a,f), cl(a), cl(c), on(c,b), on(b,f), bl(a), bl(b), bl(c) \,/ \\ &\, move(a,c) \,/\, on(a,c), \mathbf{cl(f)}, \neg cl(c), \neg on(a,f) \end{aligned}$$

The (action,effects) of this example can be post-matched with r_1, yielding the following (incorrect) generalization

$$\begin{aligned} r_g : &\, cl(X), cl(Y), on(X,Z), bl(Y), bl(X) \,/ \\ &\, move(X,Y) \,/\, on(X,Y), \neg cl(Y), \neg on(X,Z), cl(Z) \end{aligned}$$

while r_2 has to be specialized as it pre-matches the noisy example while not predicting the observed effects.

2.2 Conservative Generalizations and Specializations

Unlike in the deterministic case, examples may contradict each other, and a rule should not be specialized as soon as it is contradicted a given example (that may be noisy). Noisy examples may also induce over-generalizations. Thus, specializations and generalizations should be "cautious" and conservative. Therefore, we propose in the following an algorithm that delays actual generalizations and specializations until sufficient evidence has been collected. To that end, we attach basic estimates to each rule. They are combined to help decision making about when to generalize/specialize. To each rule r in T, we associate three basic estimates:

– $r.n_{sa}$ the number of examples pre-matched by the rule since its creation
– $r.n_{ae}$ the number of post-matched examples
– $r.n_{sae}$ the number of covered examples

Initial values for $r.n_{sa}$, $r.n_{ae}$ and $r.n_{sae}$ is 1. $(r.n_{ae} - r.n_{sae})$ is the number of times a rule post-matched an example without pre-matching it: if the rule was requested to predict (it wasn't because it is too specific), it would have predicted well. If this value is high *wrt* $r.n_{ae}$, it means that the rule could probably be generalized. The generalization trend $r.gen \in [0, 1]$ of a rule is defined as $\frac{r.n_{ae} - r.n_{sae}}{r.n_{ae}}$. A rule is generalized with an example only if $r.gen > \theta_{gen}$, where θ_{gen} is a threshold parameter of the system.. In addition, such a modification is decided only if the rule has been sufficiently evaluated, *ie* $r.n_{ae} > \theta_{evl}$, where θ_{evl} is another parameter of the system.

Similarly, a rule should be specialized if it doesn't prove to be accurate enough to predict well. Rule accuracy $r.acc$ is defined as $\frac{r.n_{sae}}{r.n_{sa}}$. If accuracy drops below a given threshold θ_{spc}, and if $r.n_{sa} > \theta_{evl}$, then r is specialized.

2.3 Covering and Elimination of Irrelevant Rules

When a new example x is not covered by any rule, the algorithm updates T to make it complete. If no relevant generalization is identified, the example is simply added as a rule in the rule set, without ancestors.

This process ensures covering all examples including noisy ones, and may introduce many irrelevant rules that might be difficult to generalize, and that the algorithm should be able to identify and delete.

So as to limit the complexity of the model, we bound the number of rules in T with a parameter N. Any new but supernumerary rule replaces another among most untrusted ones. To that end, a confidence estimate $r.cnf$ is associated with each rule. To be trusted, a rule should have been evaluated often enough, and its accuracy should be high. Therefore, $r.cnf = 0$ if $r.n_{sa} < \theta_{evl}$, and is equal to $r.acc$ otherwise.

2.4 Prediction with the Model

The above mechanisms allow for contradictions between rules. As a result, when an example is provided for prediction, more than one rule may pre-match the current state and action. Among these rules, only one among the most confident ones is used to compute the predicted effect.

3 Empirical Study

For the experimental study, we use a blocks world, used as a benchmark in most RRL works. States and actions are described with literals and objects. Objects are either blocks (a, b ...) or the floor f. Predicates $on/2$ and $cl/1$ are used to describe the blocks layout. A predicate $bl/1$ states whether an object is a block. A block is moved on top of an object using the action predicate $move/2$.

Examples are generated in sequence : each episode stops after 10 time steps or when the goal is reached (stacking all blocks), sequential actions are randomly chosen. Every ten learning examples, twenty random examples are generated and used to estimate the accuracy of the model. The *error* is the percentage of unpredicted examples (at least one effect literal is uncorrectly predicted). Results are averaged over five experiments.

So as to introduce indeterminism, a perception noise is added : the states as observed by the system may be randomly altered. With probability ϵ, random predicates are added to or removed from the state. The number of additions/removals is randomly chosen between 1 and n_ϵ. Since the examples are provided in sequence, any noisy state s_t affects two examples : x_t and x_{t+1}. This kind of noise offers a high variety of possible irrelevant observations. This perceptual noise is different from the action-noise studied in [9] where it only consists in sometimes letting a block drop onto the table instead of getting stacked.

Figure 3 shows the evolution of the error along time steps, when the number of blocks grows. The amount of noise is fixed : $\epsilon = 10\%$. Figure 4 shows the evolution of the error along time steps, when the noise grows. The number of blocks is here fixed to 6. System parameters are $\theta_{gen} = \theta_{spc} = 0.9, \theta_{evl} = 3$ and $N = 20$ (max number of rules). Environment parameter n_ϵ is 2.

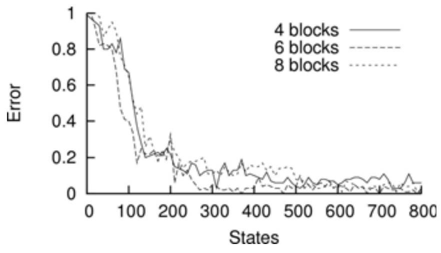


Fig. 3. 10% noise: 4, 6 and 8 blocks

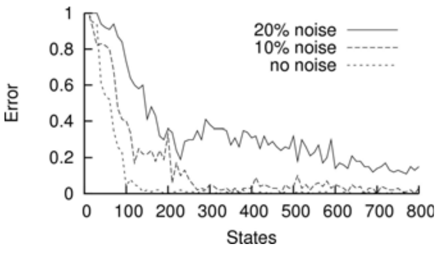


Fig. 4. 6 blocks: 0, 10 and 20% noise

The irregularity of the curves is due to the number of random samples used for evaluation *wrt* the size of the environments. Thanks to good generalization capabilities, error appears loosely dependent on the problem size. Our method is quite resistant to high level of noise, but the convergence speed drops with 20% noise. Indeed, with such an amount of noise, it is highly probable that when an action is observed, both initial and resulting states are corrupted. Together with a noise affecting several literals, very misleading and unlikely examples are presented for learning.

4 Conclusion

We have proposed in this paper an algorithm that tackles both problems of noise and full-incrementality for action model learning. Only very few examples are stored and the model might be revised for each new example. Unlike the deterministic case, this work is based on heuristics rather than careful enumerations. Simple estimates help to perform conservative and delayed generalizations and specializations of rules. The presented system proves to be efficient even with a fairly large amount of perception noise, and when the size of the problem grows. Future work will aim at proposing a fully incremental rule based system for regression, so as to approximate value-functions.

References

1. Benson, S.: Inductive learning of reactive action models. In: ICML 1995, pp. 47–54 (1995)
2. Croonenborghs, T., Ramon, J., Blockeel, H., Bruynooghe, M.: Online learning and exploiting relational models in reinforcement learning. In: IJCAI, pp. 726–731 (2007)
3. Driessens, K., Ramon, J.: Relational instance based regression for relational reinforcement learning. In: ICML, pp. 123–130 (2003)
4. Driessens, K., Ramon, J., Blockeel, H.: Speeding up relational reinforcement learning through the use of an incremental first order decision tree algorithm. In: Flach, P.A., De Raedt, L. (eds.) ECML 2001. LNCS (LNAI), vol. 2167, pp. 97–108. Springer, Heidelberg (2001)
5. Dzeroski, S., De Raedt, L., Driessens, K.: Relational reinforcement learning. Machine Learning 43, 7–52 (2001)
6. Esposito, F., Semeraro, G., Fanizzi, N., Ferilli, S.: Multistrategy theory revision: Induction and abduction in INTHELEX. Machine Learning 38(1-2), 133–156 (2000)
7. Gil, Y.: Learning by experimentation: Incremental refinement of incomplete planning domains. In: ICML, pp. 87–95 (1994)
8. Li, L., Littman, M.L., Walsh, T.J.: Knows what it knows: a framework for self-aware learning. In: ICML, pp. 568–575 (2008)
9. Pasula, H.M., Zettlemoyer, L.S.: Kaelbling L. Learning symbolic models of stochastic domains. Journal of Artificial Intelligence Research (JAIR) 29, 309–352 (2007)
10. Pasula, H.M., Zettlemoyer, L.S., Pack Kaelbling, L.: Learning probabilistic planning rules. In: ICAPS, pp. 146–163 (2004)
11. Rodrigues, C., Gérard, P., Rouveirol, C., Soldano, H.: Incremental learning of relational action rules. In: ICMLA. IEEE Computer Society, Los Alamitos (2010) (to appear)
12. Shen, W.M.: Discovery as autonomous learning from the environment. Machine Learning 12(1-3), 143–165 (1993)
13. Sutton, R.S.: Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: ICML, pp. 216–224 (1990)
14. Van Otterlo, M.: The logic of adaptive behavior. PhD thesis, University of Twente, Enschede (2008)
15. Walsh, T.J., Littman, M.L.: Efficient learning of action schemas and web-service descriptions. In: AAAI, pp. 714–719 (2008)
16. Walsh, T.J., Szita, I., Diuk, M., Littman, M.L.: Exploring compact reinforcement-learning representations with linear regression. In: UAI, pp. 714–719 (2009)
17. Wang, X.: Learning by observation and practice: An incremental approach for planning operator acquisition. In: ICML, pp. 549–557 (1995)
18. Yang, Q., Wu, K., Jiang, Y.: Learning action models from plan examples using weighted max-sat. Artificial Intelligence 171(2-3), 107–143 (2007)

When Does It Pay Off to Use Sophisticated Entailment Engines in ILP?

José Santos and Stephen Muggleton

Department of Computing, Imperial College London
{jcs06,shm}@doc.ic.ac.uk

Abstract. Entailment is an important problem in computational logic particularly relevant to the Inductive Logic Programming (ILP) community as it is at the core of the hypothesis coverage test which is often the bottleneck of an ILP system. Despite developments in resolution heuristics and, more recently, in subsumption engines, most ILP systems simply use Prolog's left-to-right, depth-first search selection function for SLD-resolution to perform the hypothesis coverage test.

We implemented two alternative selection functions for SLD-resolution: smallest predicate domain (SPD) and smallest variable domain (SVD); and developed a subsumption engine, Subsumer. These entailment engines were fully integrated into the ILP system ProGolem.

The performance of these four entailment engines is compared on a representative set of ILP datasets. As expected, on determinate datasets Prolog's built-in resolution, is unrivalled. However, in the presence of even little non-determinism, its performance quickly degrades and a sophisticated entailment engine is required.

Keywords: Entailment engines, Coverage testing, SLD-resolution.

1 Introduction and Motivation

Inductive Logic Programming (ILP) systems construct hypotheses from a rich hypothesis language and thus have to traverse a large hypothesis search space. This search requires having to test some metric of the candidate hypothesis on the provided examples. A metric typically used is coverage: positive examples covered minus negative examples covered. Evaluating coverage of a single candidate hypothesis requires thus, potentially, testing the coverage of the candidate clause on all training examples. Moreover, each one of these coverage tests can be very expensive to compute as it is a query in first-order logic.

This problem is well known to ILP researchers and several techniques have been proposed to alleviate it. Just to name a few, these techniques range from combining queries in query packs [1] to take advantage of the similar structure of the candidate clauses, transforming the clause before execution [4] so that the transformed clause is more efficient to evaluate, improving the indexing mechanism [3] of the Prolog engine, to stochastic estimation of the clause coverage [14], [7].

Another approach to improve the coverage test efficiency is to use a custom resolution engine instead of Prolog's left-to-right, depth-first search implementation of SLD-resolution. In the 1980's there was extensive research on this subject. In [15] it is noted that Prolog's default evaluation order of goals in a clause can lead to intolerable inefficiencies. The authors, motivated by a AI planning application, propose a "cheapest-first" heuristic that is akin to ours smallest predicate domain, letting the resolution engine choose, during evaluation, the predicate that has fewer solutions. They also recognize the potential overhead of this re-ordering procedure Later, [10] even proposes a machine learning approach to automatically decide the goal order in a query.

In some scenarios a full resolution engine is not needed (see section 2.1 for a discussion) and one can do the coverage test with a θ-subsumption engines. Subsumption engines optimized to perform subsumption efficiently on complex non-determinate clauses, have been developed recently, e.g. Django [9], Resumer2 [8] and Subsumer [13].

In section 2.2 we present two alternative heuristics for SLD-resolution. These heuristics, together with the subsumption engine Subsumer and Prolog's built-in SLD-resolution were integrated in the ILP system ProGolem [11]. By empirically evaluating the performance of these entailment engines on a representative set of ILP problems, we try to characterize the properties an ILP problem must have for it to pay off to use sophisticated entailment engines.

This characterization has immediate applicability to ProGolem as it already implements these four entailment engines and could choose dynamically the most suitable algorithm for each pair $\langle hypothesis, example\rangle$.

2 The θ-Subsumption Problem

θ-subsumption is an incomplete approximation to logical implication [12]. While implication is undecidable in general, θ-subsumption is a NP-complete problem [5]. A clause C θ-subsumes a clause D ($C \vdash_\theta D$) if and only if there exists a substitution θ such that $C\theta \subseteq D$.

Example 1. $C : h(X_0) \leftarrow l1(X_0, X_1), l1(X_0, X_2), l1(X_0, X_3), l2(X_1, X_2), l2(X_1, X_3)$
$D : h(c_0) \leftarrow l1(c_0, c_1), l1(c_0, c_2), l2(c_1, c_2)$
$C\theta$ subsumes D with $\theta = \{X_0/c_0, X_1/c_1, X_2/c_2, X_3/c_2\}$.

The θ-subsumption problem is thus, given two clauses, C and D, find a substitution θ such that all literals of C can be mapped into a subset of the literals of D.

Prolog performs entailment using SLD-resolution [6] which is, in general, stronger than pure subsumption (see 2.1). Within SLD-resolution all mappings from the literals in C onto the literals in D (for the same predicate symbol) are constructed left-to-right in a depth-first search manner.

As all Prolog programmers know, the order of the literals in C has a significant impact on the (in)efficiency of the query evaluation.

2.1 Subsumption versus Resolution

Selective Linear Definite clause resolution (SLD-resolution) is the inference rule in logic programming. It allows the Prolog interpreter to derive all logical consequences of a query. In order to use subsumption to decide if an example e is covered by a clause C, one needs to encode all literals related to that example in a single saturated clause S_e (see below). When used to implement ILP's coverage test, θ-subsumption generates the same solutions as SLD-resolution when the underlying prolog program (i.e. background knowledge in the ILP setting) is pure Prolog.

If the background knowledge contains non-pure Prolog constructs (e.g. non-constructive arithmetic operators, cuts, ...) subsumption will only find a subset, usually empty, of the solutions that SLD-resolution finds.

Unfortunately many real-world ILP datasets express their background knowledge in non-pure Prolog. Often the problem lies with real number arithmetic. For instance, consider the program in Figure 1.

```
:- modeh(1, active(+molecule)).          active(mol1).  logp(mol1, 3.14).
:- modeb(1, logp(+molecule,-real)).      gteq(X, X):- !.
:- modeb(1, gteq(+real,#real)).          gteq(X, Y):- X>=Y.
```

Fig. 1. Simple ILP program with non-pure background knowledge

The saturated clause for $active(mol1)$ is $active(mol1) \leftarrow logp(mol1, 3.14)$, $gteq(3.14, 3.14)$. Suppose now we have an hypothesis $active(X) \leftarrow logp(X, Y)$, $gteq(Y, 3.05)$. This hypothesis does not subsume the ground bottom clause as there is no literal $gteq(3.14, 3.05)$ in it. However, were we to use SLD-resolution we would be able to prove the hypothesis with the binding $X = mol1, Y = 3.14$.

The culprit of the problem is that the ground bottom clause did not capture the full information available in the $gteq/2$ clause. There are two problems, the cut in the first $gteq/2$ clause prevents retrieving more solutions, to the ground bottom clause of $mol1$, when the second argument is unbound or equal to the first argument. The most serious problem is that the $>=$ comparison operator is not constructive. That is, $>= /2$ requires both arguments to be instantiated, not returning in backtracking numbers that verify the condition when one or both of the arguments are unbound.

In situations like these one cannot use a θ-subsumption engine but need instead a resolution engine.

Throughout this paper we sometimes use the terms entailment, subsumption and resolution almost interchangeably although they are not equivalent. This abuse of terminology is justified because, for the purpose of our experiments, those expressions are equivalent. From the perspective of an ILP system, what matters is whether a clause covers (i.e. entails) an example or not. Both subsumption and resolution engines perform this entailment test with the same result as long as the background knowledge is pure Prolog.

2.2 Entailment Algorithms in ProGolem

ProGolem implements four entailment engines. Three are variants of SLD-resolution and one is the subsumption engine Subsumer described in [13]. The three variants of SLD-resolution are Prolog's built-in left-to-right depth-first search heuristic for SLD-resolution (hereafter Left-to-right) and two alternative selection functions for SLD-resolution. Smallest Predicate Domain (SPD-resolution for simplicity) and Smallest Variable Domain (SVD-resolution).

In SPD-resolution the literal with fewest number of solutions at each moment is picked. Note that in Prolog the literals are always picked left-to-right in the order given in the clause. This is the same as the "cheapest-first" heuristic described in [15].

SVD-resolution is more sophisticated, it computes the consistent domains of each variable and at each moment binds the variable with smallest domain with one of its possible values.

Subsumer improves upon SVD-resolution by decomposing a clause dynamically in independent components but is no longer a resolution engine. It is a θ-subsumption engine requiring the subsumee clause (the example) to be given as a ground bottom clause. Note that in this case the background knowledge is only used once to create the ground bottom clauses and is never called during a subsumption test (see Section 2.1).

2.3 Time Complexity

Let N be the length of an hypothesis H and M be the length of an example E. The worst case complexity of SLD/SPD-resolution is $O(M^N)$ as we need to map each literal of H (ranging from $1..N$) to a literal in E (ranging from $1..M$).

In practice, since the SLD/SPD-resolution tests the consistency of the matching while constructing the substitution (thus bounding other variables) and not just at the end, for clauses C with too many literals (i.e. $M \approx N$) the subsumption problem may become overconstrained and thus be easier than when M is a fraction of N.

An alternative way, employed by SVD-resolution and Subsumer, to tackle the subsumption problem is to map variables of H to terms in E rather than literals to literals. Let V be the set of distinct variables in H and T the set of distinct terms in E. We can map the θ-subsumption problem to the problem of finding a mapping from V to T. This approach has worst case complexity $O(|T|^{|V|})$.

However, it is not easy in practice to antecipate whether the literal or variable mapping works better as the average case complexity depends essentially on how constrained the search gets when a literal or a variable is bound. Note that when we map a literal all its variables get bound at the same time.

3 Empirical Evaluation

In this section we extensively compare the four entailment engines described in Section 2.2. We have not used Django or Resumer2 as it would not be practical and, as shown in [13], Subsumer is a good representative of sophisticated subsumption engines. It outperforms Django and is competitive with Resumer2. ProGolem with all the datasets and scripts to replicate these experiments can be found at: http://www.doc.ic.ac.uk/∼jcs06/papers/ilp10.

3.1 Materials and Methods

The ProGolem ILP system [11] was used with a representative set of well-known ILP datasets to generate hypotheses. Datasets PT.02, PT.15 and PT.31 are less known. These are problems 02, 15 and 31 of the Phase Transition (PT) framework [2], representing instances from the Yes, No and PT regions.

ProGolem is a bottom-up ILP engine that, among many other settings, allows the user to choose which entailment engine to use. Since we wanted to use a θ-subsumption engine, only pure Prolog was allowed in the background knowledge. That meant removing or disabling cuts and non-constructive arithmetic operators in some of datasets' (e.g. mutagenesis) background knowledge.

For the resolution algorithms the examples are provided in the background knowledge as usual in ILP. For the subsumption engine each example is a single (saturated) clause with all facts known to be true about it. Below is a small excerpt of a ground bottom clause for the mutagenesis dataset. The full clause has 77 literals.

$$\begin{aligned} active(d112) \leftarrow & atm(d112, d112\_9, h, 3, 0.136), atm(d112, d112\_8, h, 3, 0.136), \ldots \\ & atm(d112, d112\_1, c, 22, -0.125), bond(d112, d112\_6, d112\_9, 1), \ldots \\ & bond(d112, d112\_1, d112\_7, 1), bond(d112, d112\_1, d112\_2, 7). \end{aligned}$$

Table 1 summarizes important statistics on the datasets used. The columns are: number of examples, average example length, average number of distinct predicate symbols per example, average number of solutions per predicate symbol (assuming its input variables are bound) and average number of distinct terms per example. The latter four columns have the respective standard error associated. The figures in Table 1 were generated by computing the full ground clauses for each example in each dataset.Recall is the maximum number of alternative solutions a predicate may return.

As can be seen from Table 1, from the eight datasets selected, three are highly non-determinate (PT.XX) with exactly 100 solutions per distinct predicate symbol. Datasets Alzheimers-amine, Proteins and Pyrimidines are non-determinate with each predicate symbol having at most one solution. Carcinogenesis and Mutagenesis have a medium degree of non-determinism.

ProGolem was also used to induce theories for these datasets with all the intermediate hypotheses being collected to be later evaluated by the different entailment engines. When inducing theories, ProGolem's recall was set to 20. This is to limit the complexity of the hypotheses generated.

Since ProGolem is a bottom-up ILP system it is biased towards generating longer clauses. However, because some of these datasets are rather simple and all hypotheses were collected (including ones after negative reduction), many short hypotheses were generated as well. Many of those could have been generated

Table 1. Relevant statistics for the examples used per dataset

Dataset	$\|Ex\|$	Examples Len.	Pred. Symb.	Sols per P. S.	Terms per Ex.
Alz-amine	686	31±0	20±0	1±0	23±0
Carcinogenesis	298	115±4	11±0	5±1	54±1
Mutagenesis	188	83±2	2±0	41±2	48±1
Proteins	2028	287±1	42±0	1±0	36±0
Pyrimidines	2788	50±0	10±0	1±0	22±0
PT.02	400	701±0	7±0	100±0	20±0
PT.15	400	1503±1	15±0	100±0	39±0
PT.31	400	804±0	8±0	100±0	28±0

Table 2. Relevant statistics for the hypothesis used per dataset

Dataset	$\|Hyps\|$	Hypotheses Len.	Pred. Symb.	Lits per P. S.	Terms per Hyp.
Alz-amine	328	28±1	18±0	1±0	21±0
Carcinogenesis	161	43±3	6±0	4±0	29±2
Mutagenesis	382	43±1	2±0	21±1	33±0
Proteins	464	75±3	19±0	3±0	21±0
Pyrimidines	1730	42±0	10±0	4±0	32±0
PT.02	444	131±8	5±0	24±0	20±0
PT.15	68	163±32	7±1	25±1	36±1
PT.31	156	119±13	5±0	23±0	27±0

by a classical top-down ILP system like Aleph or Progol. For instance, one of the simpler hypothesis generated for the mutagenesis dataset was $active(A) \leftarrow bond(A, B, C, 1), bond(A, C, D, 2)$.

Table 2 summarizes the information on the hypotheses collected. The columns have an identical meaning to Table 1 except that column "Literals per Predicate Symbol" is the average number of times a given (distinct) predicate symbol appears on the hypothesis. Note that in a hypothesis the terms are usually variables and not just constants or function symbols.

3.2 Results and Discussion

Each entailment engine was used to test the Boolean coverage of a random sample of 20.000 pairs $\langle hypothesis, example\rangle$ from each dataset. Table 3 presents the average times, with respective standard errors, in milliseconds, per subsumption test. Whenever the subsumption test required more than 5 seconds it was aborted. The "Timeout" column has the percentage of subsumption tests in these circumnstances. To compute the average time all the timed out tests were ignored.

Table 3 shows large differences in the entailment test costs on the non-determinate datasets. On the determinate datasets Prolog's left-to-right implementation of SLD-resolution is unrivalled but the time required by SPD-resolution is still competitive. As the degree of non-determinism grows, so does

Table 3. Entailment average times with respective standard error, in ms, per dataset per entailment engine

Dataset	Entailment engines							
	Left-to-right		SPD-resolution		SVD-resolution		Subsumer	
	Avg time	Timeout	Avg time	Timeout	Avg time	Timeout	Avg time	Timeout
Alz-amine	0.0±0.0	0.00%	0.1±0.0	0.00%	0.3±0.0	0.00%	0.9±0.0	0.00%
Carcinogenesis	3.2±0.5	0.45%	0.5±0.1	0.01%	0.8±0.1	0.00%	1.8±0.3	0.01%
Mutagenesis	224±5.0	36.9%	19±1.4	0.27%	35±1.8	0.74%	9.9±0.8	0.03%
Proteins	0.1±0.0	0.00%	0.4±0.0	0.00%	21±0.1	0.00%	8.8±0.0	0.00%
Pyrimidines	0.1±0.0	0.00%	0.2±0.0	0.00%	0.3±0.0	0.00%	1.9±0.0	0.00%
PT.02	1987±9.6	98.8%	721±8.0	25.8%	421±6.3	8.53%	26±0.3	0.00%
PT.15	771±8.9	97.7%	360±6.2	64.3%	327±6.1	60.3%	142±2.5	0.37%
PT.31	2289±9.9	98.8%	405±6.1	52.1%	543±7.6	43.3%	76±1.4	0.03%

the advantage of Subsumer compared with the other entailment engines. It is important to note that Subsumer rarely timed out. However Subsumer's main drawback is its overhead on the determinate datasets and being unable to handle non-pure background knowledge.

4 Conclusions and Future Directions

Prolog's built-in left-to-right, depth-first search selection function for SLD-resolution is unrivalled on determinate datasets. However, when the dataset is even mildly non-determinate, Prolog's built-in resolution should not be used as the performance rapidly degrades and a large fraction of the entailment tests time out. For medium to highly non-determinate datasets Subsumer should be used. However, Subsumer is only applicable if the background knowledge is pure Prolog. If that is not the case then SPD-resolution should be employed.

These conclusions are not specific to ProGolem. They are valid for top-down ILP systems as well. Therefore it would be beneficial to integrate at least SPD-resolution and Subsumer in other ILP systems, e.g. Aleph.

It could be interesting to study if there are performance gains in using a specific entailment engine per pair $\langle hypothesis, example\rangle$ or whether looking at global properties of the dataset is enough to choose the best engine.

To fully take advantage of these powerful entailment engines on complex non-determinate problems such as the Phase Transition framework [2] one needs to improve the search control strategy of the ILP system. Being able to explore complex hypotheses is a necessary condition but is only half the way to enable ILP systems to learn theories on complex non-determinate domains.

Acknowledgments

The first author thanks Wellcome Trust for his Ph.D. scholarship. The second author thanks the Royal Academy of Engineering and Microsoft for funding his present 5 year Research Chair. We are indebted to three anonymous referees for valuable comments.

References

1. Blockeel, H., Dehaspe, L., Demoen, B., Janssens, G., Ramon, J., Vandecasteele, H.: Improving the efficiency of Inductive Logic Programming through the use of query packs. J. Artif. Intell. Res. (JAIR) 16, 135–166 (2002)
2. 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)
3. Santos Costa, V., Sagonas, K.F., Lopes, R.: Demand-Driven Indexing of Prolog Clauses. In: Dahl, V., Niemelä, I. (eds.) ICLP 2007. LNCS, vol. 4670, pp. 395–409. Springer, Heidelberg (2007)
4. Santos Costa, V., Srinivasan, A., Camacho, R., Blockeel, H., Demoen, B., Janssens, G., Struyf, J., Vandecasteele, H., Van Laer, W.: Query transformations for improving the efficiency of ILP systems. Journal of Machine Learning Research 4, 465–491 (2003)
5. Kapur, D., Narendran, P.: Np-completeness of the set unification and matching problems. In: Siekmann, J.H. (ed.) CADE 1986. LNCS, vol. 230, pp. 489–495. Springer, Heidelberg (1986)
6. Kowalski, R.A., Kuehner, D.: Linear resolution with selection function. Artif. Intell. 2(3/4), 227–260 (1971)
7. Kuzelka, O., Zelezný, F.: Fast estimation of first-order clause coverage through randomization and maximum likelihood. In: Cohen, W.W., McCallum, A., Roweis, S.T. (eds.) ICML. ACM International Conference Proceeding Series, vol. 307, pp. 504–511. ACM, New York (2008)
8. Kuzelka, O., Zelezný, F.: A restarted strategy for efficient subsumption testing. Fundam. Inform. 89(1), 95–109 (2008)
9. Maloberti, J., Sebag, M.: Fast theta-subsumption with constraint satisfaction algorithms. Machine Learning 55(2), 137–174 (2004)
10. Markovitch, S., Scott, P.D.: Automatic ordering of subgoals - a machine learning approach. In: NACLP, pp. 224–240 (1989)
11. Muggleton, S., Santos, J., Tamaddoni-Nezhad, A.: ProGolem: A system based on relative minimal generalisation. In: De Raedt, L. (ed.) ILP 2009. LNCS, vol. 5989, pp. 131–148. Springer, Heidelberg (2010)
12. Alan Robinson, J.: A machine-oriented logic based on the resolution principle. J. ACM 12(1), 23–41 (1965)
13. Santos, J., Muggleton, S.: Subsumer: A Prolog theta-subsumption engine. In: Technical communications of the 26th Int. Conference on Logic Programming, Leibniz International Proc. in Informatics, Edinburgh, Scotland (2010)
14. Sebag, M., Rouveirol, C.: Tractable induction and classification in first order logic via stochastic matching. In: IJCAI, vol. (2), pp. 888–893 (1997)
15. Smith, D.E., Genesereth, M.R.: Ordering conjunctive queries. Artif. Intell. 26(2), 171–215 (1985)

Stochastic Refinement

Alireza Tamaddoni-Nezhad and Stephen Muggleton

Department of Computing, Imperial College London
{atn,shm}@doc.ic.ac.uk

Abstract. The research presented in this paper is motivated by the following question. How can the generality order of clauses and the relevant concepts such as refinement be adapted to be used in a stochastic search? To address this question we introduce the concept of stochastic refinement operators and adapt a framework, called stochastic refinement search. In this paper we introduce stochastic refinements of a clause as a probability distribution over a set of clauses. This probability distribution can be viewed as a prior in a stochastic ILP search. We study the properties of a stochastic refinement search as two well known Markovian approaches: 1) Gibbs sampling algorithm and 2) random heuristic search. As a Gibbs sampling algorithm, a stochastic refinement search iteratively generates random samples from the hypothesis space according to a posterior distribution. We show that a minimum sample size can be set so that in each iteration a consistent clause is generated with a high probability. We study the stochastic refinement operators within the framework of random heuristic search and use this framework to characterise stochastic search methods in some ILP systems. We also study a special case of stochastic refinement search where refinement operators are defined with respect to subsumption order relative to a bottom clause. This paper also provided some insights to explain the relative advantages of using stochastic lgg-like operators as in the ILP systems Golem and ProGolem.

1 Introduction

Most ILP systems are traditionally based on clause refinement through a lattice defined by a generality order (e.g. subsumption). However, there is a long-standing and increasing interest in stochastic search methods in ILP for searching the space of candidate clauses (e.g. [11,15,12,18,10,8]). The research presented in this paper is motivated by the following question. How can the generality order of clauses and the relevant concepts such as refinement be adapted to be used in a stochastic search? To address this question we introduce the concept of stochastic refinement operators and adapt a framework, called stochastic refinement search.

Refinement of a clause is defined as a set of clauses. In this paper we introduce stochastic refinement of a clause as a probability distribution over a set of clauses. This probability distribution can be viewed as a prior in a stochastic ILP search. In this paper we define the concept of stochastic refinement search. In general a stochastic refinement search can be viewed as a Markov chain. We study the properties of a stochastic refinement search as two well known Markovian approaches: 1) Gibbs sampling algorithm and 2) random heuristic search. As a Gibbs sampling algorithm, a stochastic refinement search iteratively generates random samples from the hypothesis space according to a posterior distribution. We have shown that a minimum sample size can be set so that in each iteration a consistent clause is generated with a high probability.

We define a special case of random heuristic search [17] called monotonic random heuristic search and then we show that due to the generality order defined by the refinement operators, a stochastic refinement search can be viewed as a monotonic random heuristic search. The advantage of studying stochastic refinement search as a random heuristic search is that we can use the theoretical results from random heuristic search in order to analyse the behaviour and convergence of the search. We also study a special case of stochastic refinement search where refinement operators are defined with respect to subsumption order relative to a bottom clause [16]. This paper also provides some theoretical insights to explain the relative advantages of using stochastic lgg-like operators as in the ILP systems Golem and ProGolem.

This paper is organised as follows. In Section 2 we review some of the basic concepts from ILP which are used in the definitions and theorems in this paper. In Section 3 stochastic refinement operators are introduced and their properties are discussed. The framework of stochastic refinement search is discussed in Section 4 and this framework is used to characterise stochastic search methods in some ILP systems. In Section 5 we consider a special case where a stochastic search is used to explore a refinement graph bounded by a most specific (bottom) clause. Section 6 concludes the paper.

2 Preliminaries

We assume the reader to be familiar with the basic concepts from logic programming and inductive logic programming [9]. This section is intended as a brief reminder of some of the concepts used in definitions and theorems in this paper.

The general subsumption order on clauses, also known as θ-subsumption, is defined as follows.

Definition 1 (Subsumption). *Let C and D be clauses. We say C subsumes D, denoted by $C \succeq D$, if there exists a substitution θ such that $C\theta$ is a subset of D. C properly subsumes D, denoted by $C \succ D$, if $C \succeq D$ and $D \not\succeq C$. C and D are subsume-equivalent, denoted by $C \sim D$, if $C \succeq D$ and $D \succeq C$.*

Remark 1. Let $\mathcal{C}$ be a set of first order clauses and $\succeq$ be the subsumption order as defined in Definition 1. Every finite subset of $\mathcal{C}$ has a *most general specialisation (mgs)* and a *least general generalisation (lgg)*. Thus $\langle \mathcal{C}, \succeq \rangle$ is a lattice.

The following definition is a reminder of the concept of unary refinement operators and related properties.

Definition 2 (Unary refinement operator). *Let $\langle G, \succeq \rangle$ be a quasi-ordered set. A (downward)* unary refinement operator *for $\langle G, \succeq \rangle$ is a function $\rho : G \rightarrow 2^G$, 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 $\rho^*(C) = \rho^1(C) \cup \rho^2(C) \cup ..$*
- *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* 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 unary refinement operator.

Example 1. Figure 1 shows part of a (downward) unary refinement graph for the subsumption order. In this graph clause $p(x, y)$ is refined either by unifying variables or by adding literals. The refinement operator presented by this graph is not complete as it does not include all possible refinements. It is proper as the graph does not contain cycles. It is redundant because it does not have a tree structure and there is more than one path from $p(x, y)$ to $p(x, x) \leftarrow q(x, z)$.

The following definition for binary refinement is adapted from [5].

Definition 3 (Binary refinement operator). *Let $\langle G, \succeq \rangle$ be a quasi-ordered set. A (downward)* binary refinement operator *for $\langle G, \succeq \rangle$ is a function $\rho : G^2 \rightarrow 2^G$, such that $\rho(C, D) \subseteq \{E | C \succeq E, D \succeq E\}$, for every $C \in G$.*

- *The sets of* one-step refinements, n-step refinements *and* refinements *of some $C \in G$ are respectively: $\rho^1(C, D) = \rho(C, D)$, $\rho^n(C, D) = \{E|$ there is an $F \in \rho^{n-1}(C, D)$ and an $H \in \rho^{n-1}(C, D)$ such that $E \in \rho(F, H)\}, n \geq 2$ and $\rho^*(C, D) = \rho^1(C, D) \cup \rho^2(C, D) \cup ..$*
- *A ρ-*chain *(C, D) to E is a sequence $(C, D) = (C_0, D_0), (C_1, D_1), \ldots, (C_m, D_m)$, such that $E = C_m$ or $E = D_m$ and $C_i, D_i \in \rho(C_{i-1}, D_{i-1})$ for every $1 \leq i \leq m$.*
- *ρ is* locally finite *if for every $C, D \in G$, $\rho(C, D)$ is finite and computable.*
- *ρ is* proper *if for every $C, D \in G$, $\rho(C, D) \subseteq \{E | C \succ E, D \succ E\}$.*
- *ρ is* complete *if for every $B, C, D \in G$ such that $C \succ B$, $D \succ B$ there is an $E \in \rho^*(C, D)$ such that $B \sim E$ (i.e. B and E are equivalent in the $\succeq$-order).*
- *ρ is* weakly complete *for $\langle G, \succeq \rangle$ if $\rho^*(\Box, \Box) = G$, where $\Box$ is the top element of G.*

$$\begin{tikzcd}
C: & & p(x,y) \arrow[dl] \arrow[d] \arrow[dr] & & \\
\rho(C): & p(x,x) \arrow[d] & p(x,y) \leftarrow q(x,z) \arrow[dl] \arrow[d] \arrow[dr] & p(x,y) \leftarrow r(w,y) \arrow[dr] & \\
\rho^2(C): & p(x,x) \leftarrow q(x,z) \arrow[d] & p(x,y) \leftarrow q(x,x) \arrow[d] & \begin{array}{c} p(x,y) \leftarrow q(x,z), \\ r(w,y) \end{array} \arrow[d] & \begin{array}{c} p(x,y) \leftarrow r(w,y), \\ q(x,z) \end{array} \arrow[d] \\
\rho^3(C): & p(x,x) \leftarrow q(x,y) \arrow[d] & \begin{array}{c} p(x,y) \leftarrow q(x,x), \\ q(x,y) \end{array} \arrow[d] & \begin{array}{c} p(x,y) \leftarrow q(x,x), \\ r(w,y) \end{array} \arrow[d] & \begin{array}{c} p(x,y) \leftarrow r(y,y), \\ q(x,z) \end{array} \arrow[d] \\
\rho^n(C): & \dots & \dots & \dots & \dots
\end{tikzcd}$$

$$\rho^*(C) = \rho^0(C) \cup \rho^1(C) \cup \rho^2(C) \dots$$

Fig. 1. Part of a unary refinement graph representing (downward) refinements of a clause

- ρ *is* non-redundant *if for every* $B, C, D, E, F \in G$, $F \in \rho^*(B, C)$ *and* $F \in \rho^*(D, E)$ *implies* $B, C \in \rho^*(D, E)$ *or* $D, E \in \rho^*(B, C)$.
- 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 binary refinement operator.

3 Stochastic Refinement Operators

In this section we introduce the concept of stochastic refinement operators. According to Definition 2 (and Definition 3), refinement of a clause (or a pair of clauses) is a set of clauses. In the following we define stochastic refinement as a probability distribution over a set of clauses.

Definition 4 (Stochastic unary refinement operator). *Let ρ be a (downward) unary refinement operator for the quasi-ordered set $\langle G, \succeq \rangle$ and $C \in G$. A (downward)* stochastic unary refinement operator *is a function $\sigma : G \rightarrow 2^{G \times [0,1]}$ defined as follows: $\sigma(C) = \{\langle D_i, p_i \rangle | D_i \in \rho(C),\ p_i \in [0,1]$ and $\sum p_i = 1$ for $1 \leq i \leq |\rho(C)|\}$.*

- *A **σ-chain** from C to D, denoted by $C \xrightarrow{\sigma} D$, is a sequence $C = C_0, C_1, \dots, C_m = D$, such that $\langle C_i, p_i \rangle \in \sigma(C_{i-1})$ for every $1 \leq i \leq m$ and the probability of this σ-*chain *is $p(C \xrightarrow{\sigma} D) = \prod_{i=1}^{m} p_i$.*
- ***n-step stochastic refinements** of C is defined as: $\sigma^n(C) = \{\langle D, p \rangle | D \in \rho^n(C)$ and $p = \sum_{x \in X} p(x)$ where X is the set of σ-chains from C to $D\}$.*

$$\begin{tikzcd}
C: & & p(x,y) \arrow[dl] \arrow[d] \arrow[dr] & \\
\rho(C): & p(x,x) & p(x,y) \leftarrow q(x,z) & p(x,y) \leftarrow r(w,y)
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
C: & & p(x,y) \arrow[dl, "0.5"'] \arrow[d, "0.3"] \arrow[dr, "0.2"] & \\
\sigma(C): & p(x,x) & p(x,y) \leftarrow q(x,z) & p(x,y) \leftarrow r(w,y)
\end{tikzcd}$$

(b)

Fig. 2. (a) Refinement of a clause is defined as a set of clauses (b) Stochastic refinement of a clause is defined as a probability distribution over a set of clauses

– ***stochastic refinements*** *of C is defined as:* $\sigma^*(C) = \{\langle D_i, p_i\rangle | D_i \in \rho^*(C),$ $p_i \in [0,1]$ *and* $\sum p_i = 1$ *for* $1 \leq i \leq |\rho^*(C)|\}$.
– *σ inherits the standard properties (i.e. local finiteness, properness and completeness) from ρ.*

Example 2. Figure 2.a shows refinement of a clause as a set of clauses. Stochastic refinement of a clause is defined as a probability distribution over a set of clauses (Figure 2.b).

Example 3. Figure 3 shows part of a stochastic refinement graph. This graph shows the probabilities of clauses in $\sigma^n(C)$ as defined in Definition 4. For example, there are two σ-chains from C to D_4. Hence, the probability of D_4 in $\sigma^2(C)$ is $0.5 \times 1.0 + 0.3 \times 0.4 = 0.62$.

According to Definition 4, $\sigma(C)$ and $\sigma^*(C)$ represent probability distributions. In the following we show that $\sigma^n(C)$ is also a probability distribution.

Theorem 1. *n-step stochastic refinements of a clause represent a probability distribution.*

Proof. Let $\sigma^n(C)$ be n-step stochastic refinements of clause C as defined in Definition 4 such that $\sigma^n(C) = \{\langle D_1, p_1\rangle, \langle D_2, p_2\rangle, \ldots \langle D_m, p_m\rangle\}$. We will show that $\sum_{i=1}^{m} p_i = 1$. The proof is by induction on n. For $n = 1$ the theorem is true by definition of $\sigma(C)$. Assume that the theorem is true for k then the sum of probabilities $p_1, p_2, \ldots, p_s$ at level k is 1: $\sum_{i=1}^{s} p_i = 1$. Suppose that each node with probability p_i at level k is extended into t nodes with probabilities $q_{i1}, q_{i2}, \ldots, q_{it}$. Then the sum of probabilities at level $k+1$ will be: $\sum_{i=1}^{s}\sum_{j=1}^{t} p_i q_{ij} = \sum_{i=1}^{s} p_i(q_{i1} + q_{i2} + \cdots + q_{it})$. But $q_{i1} + q_{i2} + \cdots + q_{it} = 1$ and $\sum_{i=1}^{s} p_i = 1$ and therefore the sum of probabilities at level $k+1$ will be 1 and this completes the proof. $\square$

In the following we define analogous concepts for stochastic binary refinement operators.

$$\begin{tikzcd}
C: & & C \arrow[dl, "0.5"'] \arrow[d, "0.3"] \arrow[dr, "0.2"] & & \\
\sigma(C): & \langle D_1, 0.5\rangle \arrow[d, "1.0"] & \langle D_2, 0.3\rangle \arrow[dl, "0.4"'] \arrow[d, "0.4"] \arrow[dr, "0.2"] & \langle D_3, 0.2\rangle \arrow[dr, "1.0"] & \\
\sigma^2(C): & \langle D_4, 0.62\rangle \arrow[d] & \langle D_5, 0.12\rangle \arrow[d] & \langle D_6, 0.06\rangle \arrow[d] & \langle D_7, 0.2\rangle \arrow[d] \\
\sigma^n(C): & \dots & \dots & \dots & \dots
\end{tikzcd}$$

Fig. 3. Part of a stochastic refinement graph

Definition 5 (Stochastic binary refinement operator). *Let ρ be a (downward) binary refinement operator for the quasi-ordered set $\langle G, \succeq\rangle$ and $C, D \in G$. A (downward)* stochastic binary refinement operator *is a function $\sigma : G^2 \to 2^{G\times[0,1]}$ defined as: $\sigma(C, D) = \{\langle E_i, p_i\rangle | E_i \in \rho(C, D),\ p_i \in [0, 1]$ and $\sum p_i = 1$ for $1 \leq i \leq |\rho(C, D)|\}$.*

- *A **σ-chain** from (C, D) to E, denoted by $(C, D) \xrightarrow{\sigma} E$, is a sequence $(C, D) = (C_0, D_0), (C_1, D_1), \dots, (C_m, D_m)$, such that $E = C_m$ or $E = D_m$ and $\langle C_i, p_i\rangle, \langle D_i, q_i\rangle \in \sigma(C_{i-1}, D_{i-1})$ for every $1 \leq i \leq m$ and the probability of this σ-*chain *is $p((C, D) \xrightarrow{\sigma} E) = \prod_{i=1}^{m} p_i q_i$.*
- *n**-step stochastic binary refinements** of some $(C, D) \in G^2$ is defined as: $\sigma^n(C, D) = \{\langle E, p\rangle | E \in \rho^n(C, D)$ and $p = \sum_{x\in X} p(x)$ where X is the set of σ-chains from (C, D) to $E\}$.*
- ***stochastic binary refinements** of some $(C, D) \in G^2$ is defined as: $\sigma^*(C, D) = \{\langle E_i, p_i\rangle | E_i \in \rho^*(C, D),\ p_i \in [0, 1]$ and $\sum p_i = 1$ for $1 \leq i \leq |\rho^*(C, D)|\}$.*
- *σ inherits the standard properties (i.e. local finiteness, properness and completeness) from ρ.*

Example 4. Figure 4 shows stochastic binary refinement of a pair of clauses as a probability distribution over a set of clauses.

As for $\sigma^n(C)$, it can be shown that $\sigma^n(C, D)$ is a probability distribution.

Theorem 2. *n-step stochastic binary refinements of a pair of clauses represent a probability distribution.*

Proof. Let $\sigma^n(C, D)$ be n-step stochastic binary refinements of clause C and D as defined in Definition 5 such that $\sigma^n(C, D) = \{\langle D_1, p_1\rangle, \langle D_2, p_2\rangle, \dots \langle D_m, p_m\rangle\}$. We will show that $\sum_{i=1}^{m} p_i = 1$. The proof is similar to the proof of Theorem 1 except that we need to show that the sum of probabilities at level $k + 1$ will be: $\sum_{i=1}^{s} \sum_{j=1}^{t} \sum_{l=1}^{u} p_i q_j r_{ijl} = \sum_{i=1}^{s} \sum_{j=1}^{t} p_i q_j (r_{ij1} + r_{ij2} + \dots + r_{iju})$ where the binary refinement of two nodes with probabilities p_i and q_j at level k is extended into u nodes with probabilities $r_{ij1}, r_{ij2}, \dots, r_{iju}$. But $r_{ij1} + r_{ij2} + \dots + r_{iju} = 1$ and $\sum_{i=1}^{s} \sum_{j=1}^{t} p_i q_j = \sum_{i=1}^{s} p_i (q_1, q_2, \dots, p_t) = \sum_{i=1}^{s} p_i = 1$ and therefore the sum of probabilities at level $k + 1$ will be 1 and this completes the proof. $\square$

$$\begin{tikzcd}
(C,D): & & C \arrow[dl] \arrow[dr] \arrow[drrr] & & D \arrow[dlll] \arrow[dl] \arrow[dr] & \\
\sigma(C,D): & \langle E_1, 0.4\rangle & & \langle E_2, 0.4\rangle & & \langle E_3, 0.2\rangle
\end{tikzcd}$$

Fig. 4. Stochastic binary refinement of a pair of clauses is defined as a probability distribution over a set of clauses

4 Stochastic Refinement Search

Different stochastic search methods have been used to explore the space of candidate clauses in ILP. Examples of these search methods are simulated annealing (e.g.[11,13]), genetic algorithms (e.g. [15]) and randomised restarted search (e.g. [18]). In this section we define a general framework which can be used to characterise some of stochastic search methods in ILP.

Definition 6 (Stochastic refinement search). *Let G and σ be defined as in Definition 4 and S_0 be a sample from G with size s. A stochastic refinement search involves a sequence $S_0 \xrightarrow{\sigma} S_1 \xrightarrow{\sigma} S_2 \xrightarrow{\sigma} \ldots$ where S_{i+1} is generated from S_i such that for each $C_{i+1} \in S_{i+1}$ there exists $C_i \in S_i$ such that $\langle C_{i+1}, p\rangle \in \sigma(C_i)$. Similarly a stochastic refinement search can be defined for a stochastic binary refinement operator σ (Definition 5) where for each $C_{i+1} \in S_{i+1}$ there exist C_i and $D_i \in S_i$ such that $\langle C_{i+1}, p\rangle \in \sigma(C_i, D_i)$.*

Figure 5 shows stochastic refinement search with unary and binary stochastic refinement operators. In Definition 6, the initial sample S_0 corresponds to the starting point of the search as in simulated annealing or an initial population as in genetic algorithms. These are usually generated randomly. The sample size s is equal to one in simulated annealing and it is the population size in a genetic algorithm. Stochastic refinement operator σ corresponds to task-specific operators or search transition rules in an stochastic ILP search algorithm. These operators generate new clauses from the clauses in the previous search state or population. For example, in the simulated annealing search used in ILP (e.g.[11,13]), the transition operators can be viewed as downward stochastic unary refinement operators which stochastically choose the next literal to be added to the body of a given clause. Crossover operators of a genetic algorithm search used in ILP (e.g. [15]) can be viewed as stochastic binary refinement operators. Note that some stochastic refinement searches (e.g. genetic algorithms) use both stochastic unary and binary refinement operators.

According to Definition 6 a stochastic refinement search is a sequence of random samples with the property that the current state depends only on the previous state. Hence, in general a stochastic refinement search can be viewed as a Markov chain. In this section we study the properties of a stochastic refinement search as two well known Markovian approaches: 1) a Gibbs sampling algorithm and 2) a random heuristic search.

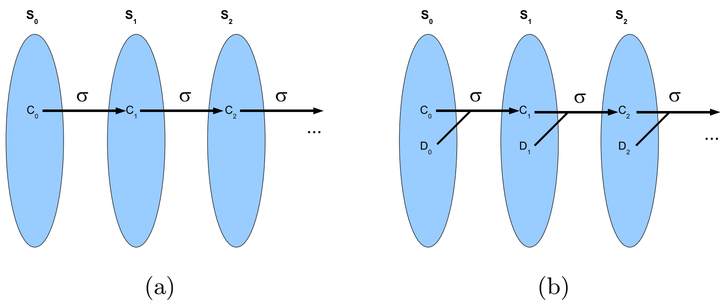


Fig. 5. A stochastic refinement search with (a) unary stochastic refinement operator (b) binary stochastic refinement operator

4.1 Stochastic Refinement Search as a Gibbs Sampling Algorithm

As noted in [8], some stochastic machine learning algorithms can be viewed as Gibbs-MAP algorithms. According to [3], most machine learning algorithms can be viewed as approximations to the following general Bayesian statistical inference algorithms.

MAP - returns the hypothesis having maximum posterior probability.
Gibbs - randomly samples hypotheses according to the posterior distribution.
Bayes Prediction - classifies unseen instances based on weighted joint prediction of the entire hypothesis space.

All of the above assume a Bayes' prior distribution over the hypothesis space. In the case of Gibbs this is used for sampling. Stochastic refinement search introduced in this paper assumes a prior distribution over the hypothesis space which is defined by stochastic refinement operators. A stochastic refinement search can be viewed as a Gibbs-MAP algorithm. A Gibbs-MAP algorithm is a Gibbs-like approximations to MAP based on sampling. It has been shown (e.g. [1]) that the sequence of samples in a Gibbs sampling algorithm constitutes a Markov chain. It has also been shown [2] that average-case error bounds for Gibbs algorithms are comparable to other Bayesian algorithms. A stochastic refinement search is a Gibbs-MAP algorithm which is aimed at maximising posterior probability by iteratively generating new samples from the hypothesis space. In a stochastic refinement search each new sample generated from the previous sample using stochastic refinement operators. There is a trade-off between the efficiency and accuracy which can be controlled by the sample size. In this section we show how with a proper sample size one could guarantee that with a high probability a consistent and compressive clause can be generated at each generation. We give examples of stochastic refinement search methods with unary and binary refinement operators.

The Quick Generalisation (QG) algorithm described in [8] is a Gibbs-MAP algorithm, which by construction, generates consistent clauses (by stochastically pruning Progol bottom clauses). A sampling algorithm based on the QG algorithm returns the clause with highest positive compression from a sample of s calls to QG. In the QG sampling algorithm described in [8], the sample size s is set by the user so that at least one of the clauses has positive compression. In this setting, the algorithm simply returns a consistent clause with the highest positive compression. The QG sampling algorithm can be viewed as a stochastic refinement search with unary refinement operator which randomly samples from "fringe" clauses (i.e. maximally general consistent clauses in the hypothesis space). As shown in [8], a minimum sample size for a QG sampling algorithm can be estimated based on the percentage of consistent clauses which have positive compression.

In the following we show that a minimum sample size can be also estimated for a stochastic refinement search with a binary operator. Golem [4] and ProGolem [6] can be viewed as stochastic refinement search methods which use lgg-like binary operators. Unlike in QG, the stochastic refinements in Golem and ProGolem do not guarantee to generate consistent clauses. However, the following theorem shows how the sample size and the probability of generating a consistent clause are related. This can be used to set a minimum sample size such that with a high probability at least one consistent lgg is generated.

Theorem 3. *Let k be an upper bound on the number of clauses in a target theory, c be a natural number and E^+ and E^- be the set of positive and negative examples respectively. Suppose that $s = ck$ pairs of clauses are randomly sampled from E^+. Then the probability that there is a pair of clauses C_1 and C_2, such that $lgg(C_1, C_2)$ is consistent with E^-, is at least $1 - e^{-c}$.*

Proof. First suppose that the k clauses in the target theory have disjoint coverage and each covers the same number of clauses. In this case, there are k partitions each covered by one clause in the target theory. The probability that a randomly selected pair of clauses belong to the same partition is $\frac{1}{k}$. If we randomly sample $s = ck$ pairs of clauses, the probability that there is no pair of clauses belonging to the same partition is $(1 - \frac{1}{k})^{ck}$. If the coverages are of different sizes then the probability that there is not a pair of clauses covered by the same clause in the target theory is less than $(1 - \frac{1}{k})^{ck}$. This is because the product of the probabilities are maximum if the coverages have the same size. Similarly, if the coverages are not disjoint then the probability that there is not a pair of clauses covered by the same clause in the target theory is less than $(1 - \frac{1}{k})^{ck}$. Also note that the maximum value for $(1 - \frac{1}{k})^{ck}$ is $\lim_{k\to\infty}(1 - \frac{1}{k})^{ck} = e^{-c}$. Then the probability that a randomly selected pair of clauses C_1 and C_2 are both covered by the same target clause is at least $1 - e^{-c}$. But if C_1 and C_2 are both covered by the same target clause C then $lgg(C_1, C_2)$ is consistent because by definition $lgg(C_1, C_2)$ is more specific than C which is consistent. Hence, the probability that there is a pair of clauses C_1 and C_2 such that $lgg(C_1, C_2)$ is consistent is at least $1 - e^{-c}$. □

Example 5. Consider the Golem algorithm as described in [4]. Suppose that the upper bound on the number of clauses in the target theory is k. Then according to Theorem 3, by selecting a minimum sample size $s = 2k$, the probability that a consistent lgg is generated is at least $1 - e^{-2} = 0.86$.

4.2 Stochastic Refinement Search as a Random Heuristic Search

It has been shown [17] that many stochastic search methods including simulated annealing, stochastic beam search and genetic algorithms are instances of a general framework called random heuristic search. The basic conditions are that the search space (Ω) be finite and that the search transition rules (τ) be Markovian and expressible as the result of s independent identically distributed random choices. The finiteness condition is not a serious limitation in most cases, including ILP, since in practice it is common to use a depth bound which leads to a finite search space. In this section we show that a stochastic refinement search is a special case of random heuristic search. First we define a random heuristic search. The following definition is adapted from [17].

Definition 7 (Random heuristic search). *Let $\Omega = \{x_0, x_1, \ldots, x_{n-1}\}$ be a search space and P_0 be a sample from Ω with size s. A random heuristic search involves a sequence $P_0 \to P_1 \to P_2 \to \ldots$ where each sample P_{i+1} is generated from previous sample P_i. Each sample P_i can be represented by a probability vector $p_i = \langle t_0, t_1, \ldots, t_{n-1} \rangle$ such that t_j is the proportion of x_j in P_i. Hence, a random heuristic search $P_0 \to P_1 \to P_2 \to \ldots$ can be also denoted as a sequence of corresponding probability distributions $p_0 \xrightarrow{\tau} p_1 \xrightarrow{\tau} p_2 \xrightarrow{\tau} \ldots$ where p_i is the probability vector for P_i and this sequence is generated by iterating a transition rule $\tau : \Delta^n \to \Delta^n$ where $\Delta^n = \{\langle s_0, s_1, \ldots, s_{n-1} \rangle | \sum_{j=0}^{n-1} s_j = 1,\ s_j \geq 0$ for all $s_j\}$.*

Example 6. Suppose in Definition 7 we have $\Omega = \{0, 1, 2, 3, 4, 5\}$ and $P_0 = \{1, 0, 3, 1, 1, 3, 2, 2, 4, 0\}$. Then, P_0 can be represented by the probability vector $p_0 = \langle 0.2, 0.3, 0.2, 0.2, 0.1, 0.0 \rangle$. Here the proportional representation given by p_0 determines the sample P_0 once the sample size is known.

Note that in Definition 7, Δ^n serves as both the space of samples of Ω and the space of probability distributions over Ω.

From Definitions 6 and 7 it is clear that a stochastic refinement search is an instance of random heuristic search. Figure 6 shows stochastic refinement search versus random heuristic search. As shown in this figure, a stochastic refinement operator σ operates on the elements of a sample S_i while a transition rule τ works directly on the corresponding sample distribution p_i. A main difference between a stochastic refinement search and a random heuristic search is that in general, a random heuristic search does not consider any ordering over Ω or for the transition rule τ. However, a stochastic refinement search is a directed search due to the generality order defined by the refinement operators. In the following we define a monotonic random heuristic search and show that an upward (or downward) stochastic refinement search can be viewed as a monotonic random heuristic search.

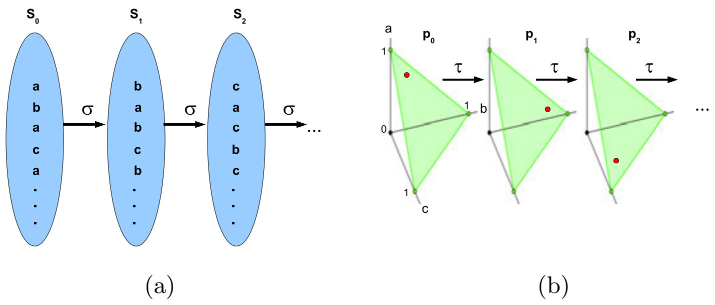


Fig. 6. (a) Stochastic refinement search (b) Random heuristic search. A stochastic refinement operator σ, operates on the elements of a sample S_i while a transition rule τ works directly on the corresponding probability vector p_i.

Definition 8 (Monotonic random heuristic search). *Let the search space Ω and the random heuristic search $P_0 \to P_1 \to P_2 \to \ldots, \ldots$ be defined as in Definition 7 and $\leq$ be a binary relation on Ω such that $\langle \Omega, \leq \rangle$ is a quasi-ordered set. If for each i, $x \in P_i$ is replaced by $x' \in P_{i+1}$ and we have $x \leq x'$ then the random heuristic search $P_0 \to P_1 \to P_2 \to \ldots$ is said to be monotonic with respect to $\leq$.*

The following Proposition follows directly from Definitions 2, 6 and 8.

Proposition 1. *A stochastic refinement search is a monotonic random heuristic search.*

Proof. According to Definitions 2 and 6, for each i, $C_i \in S_i$ and $C_{i+1} \in S_{i+1}$ we have $C_i \succeq C_{i+1}$ and therefore according to Definition 8 a stochastic refinement search is a monotonic random heuristic search with respect to $\succeq$.

The advantage of studying stochastic refinement search as a random heuristic search is that we can use the theoretical results from random heuristic search in order to analyse the behaviour and convergence of the search. In Definition 7, each state P_{i+1} of the search only depends on the previous state P_i. Hence, it can be shown that a random heuristic search can be described as a Markov chain. This property has been used [17] to estimate the probability that a particular population is generated in the next iteration. An analysis of the convergence of random heuristic search, which can also be applied to stochastic refinement search, has been discussed in [17].

4.3 Examples of Stochastic Refinement Search

In this section we discuss examples of ILP systems which use some form of stochastic refinement. In each case we characterise some aspects of the stochastic refinement with respect to the results presented in this paper.

Golem - Golem [4] is a bottom-up ILP system which employs determinate least general generalisation under relative subsumption (RLGG). Golem combines random sampling and a hill-climbing search to construct RLGGs with new randomly sampled positive examples at each iteration. The refinement in Golem can be viewed as an upward binary stochastic refinement. As shown in Example 5, with a sample size $s = 2k$ where k is the upper bound for number of clauses in the target theory, the probability that there is a pair of clauses C_1 and C_2, such that $RLGG(C_1, C_2)$ is consistent, is at least $1 - e^{-2} = 0.86$. The $RLGG$ is constructed with new sampled examples at each iteration until it converges to a consistent clause which covers a partition of positive examples.

SFoil - SFoil [11] is a top-down stochastic ILP system which combines Foil with a stochastic search in the form of simulated annealing. The stochastic search is used to choose the next literal to be added to the body of the clause. Hence, the refinement in SFoil can be viewed as downward unary stochastic refinement. The behaviour of the search can be analysed using the framework of random heuristic search. The following analysis is adopted from [17]. The sample size for simulated annealing is $s = 1$ and given a population (i.e. probability vector p) and an objective function f, the next population (i.e. probability vector q) is obtained by the following stochastic procedure: 1) sample q from a neighborhood $N(p)$ of p 2) if $f(q) < f(p)$ then the next generation is q, otherwise the next generation is q with probability $e^{(f(p)-f(q))/T_t}$ where T_t is the temperature at generation t. Then the heuristic function h which determines the stochastic transition between two distinct states i, j is defined as follows: $h(t,j)_i = \frac{[i \in N(j)]}{|N(j)|}([f(i) < f(j)] + [f(i) \geq f(j)]e^{(f(j)-f(i))/T_t})$ where $[expr]$ returns 1 if $expr$ is true and 0 otherwise.

GA-Progol - GA-Progol [15] is a stochastic ILP system in which the A^*-like search of Progol is replaced by a genetic algorithm. The stochastic refinement in GA-Progol includes both unary stochastic refinement (mutation) and binary stochastic refinement (crossover). GA-Progol also uses stochastic lgg and mgs operators relative to a bottom clause ($lgg_\perp$ and $mgs_\perp$). As for simulated annealing, a simple genetic algorithm can be characterised [17] within the framework of random heuristic search.

Aleph - Aleph [14] implements several randomised search methods including randomised local searches such as GSAT, Walksat, simulated annealing and also a randomised rapid restart search [18]. Randomised local searches in Aleph can be characterised as stochastic refinement search. Note that in some cases the stochastic refinement operators in Aleph are bi-directional and therefore the corresponding stochastic searches are non-monotonic.

QG - Quick Generalisation (QG) algorithm [8] constructs maximally general consistent clauses by stochastically pruning Progol bottom clauses. A sampling algorithm based on the QG algorithm returns the clause with highest positive compression from a sample of s calls to QG. The algorithm can be made arbitrarily efficient by choice of sample size (down to 1). The QG sampling algorithm can be viewed as a stochastic refinement search with unary refinement operator which randomly samples from “fringe” clauses (i.e. maximally general consistent clauses in the hypothesis space). A minimum sample size for a QG sampling algorithm can be estimated based on the percentage of consistent clauses which have positive compression.

ProGolem - ProGolem [6] implements an efficient (and non-determinate) variant of Golem's RLGG for the subsumption relative to $\perp$. ProGolem combines bottom-clause construction in Progol with a Golem control strategy which uses Asymmetric Relative Minimal Generalisation (ARMG) in place of determinate RLGG. ARMG in ProGolem can be viewed as stochastic binary refinement operator relative to $\perp$. Stochastic refinement relative to a bottom clause is discussed in Section 5. ProGolem combines random sampling and a beam search to construct ARMGs with new examples at each iteration.

5 Stochastic Refinement Relative to a Bottom Clause

In this section we study a special case of stochastic refinement search where a refinement space bounded by a most specific (bottom) clause is explored. This refinement space defines the search space of the ILP systems Progol and Aleph and different variants of these systems including stochastic variants described in the previous section (e.g. *QG*, ProGolem). The lattice structure and refinement operators for the subsumption order relative to $\perp$ were studied in [16]. It was shown that the refinement space of a Progol-like ILP system can be characterised using the subsumption order relative to $\perp$. It was also shown [16,6] that, unlike for the general subsumption order, efficient *lgg*-like operators can be implemented for the subsumption relative to a bottom clause (e.g. $lgg_{\perp}$ and $armg_{\perp}$). The following definitions are a summary of the subsumption order relative to $\perp$.

Definition 9 ($\overrightarrow{\mathcal{L}}_{\perp}$). *Let $\overrightarrow{\perp}$ be the bottom clause and $\overrightarrow{C}$ a definite ordered clause as defined in [16]. $\overrightarrow{\top}$ is $\overrightarrow{\perp}$ with all variables replaced with new and distinct variables. $\theta_{\top}$ is a variable substitution such that $\overrightarrow{\top}\theta_{\top} = \overrightarrow{\perp}$. $\overrightarrow{C}$ is in $\overrightarrow{\mathcal{L}}_{\perp}$ if $\overrightarrow{C}\theta_{\top}$ is a subsequence of $\overrightarrow{\perp}$.*

Definition 10 (Subsumption relative to $\perp$). *Let $\overrightarrow{\perp}$, $\theta_{\top}$ and $\overrightarrow{\mathcal{L}}_{\perp}$ be as defined in Definition 9 and $\overrightarrow{C}$ and $\overrightarrow{D}$ be ordered clauses in $\overrightarrow{\mathcal{L}}_{\perp}$. We say $\overrightarrow{C}$ subsumes $\overrightarrow{D}$ relative to $\perp$, denoted by $\overrightarrow{C} \succeq_{\perp} \overrightarrow{D}$, if $\overrightarrow{C}\theta_{\top}$ is a subsequence of $\overrightarrow{D}\,\theta_{\top}$.*

5.1 Divisibility of the Subsumption Lattice Relative to $\perp$

In this section we review the concept of a divisible lattice as described in [5] and we show that the subsumption lattice relative to $\perp$ is a divisible lattice. The following definitions are adapted from [5] for refinement relative to $\perp$.

Definition 11. *Let $\langle G, \succeq \rangle$ be a lattice, $\perp$ be the bottom element and ρ be an upward refinement operator for this lattice. Then the depth of a clause in a unary* *refinement graph, denoted by* $d^{(1)}_{\perp}$, *is defined as follows:* $d^{(1)}_{\perp}(C) = min_n[\exists D \sim C, D \in \rho^n(\perp)]$. *Similarly, the depth of a clause in a binary refinement graph, denoted by* $d^{(2)}_{\perp}$, *is defined as follows:* $d^{(2)}_{\perp}(C) = min_n[\exists D \sim C, D \in \rho^n(\perp, \perp)]$.

Definition 12. *Lattice* $\langle G, \succeq \rangle$ *is said to be divisible with respect to* ρ *if for all* $E \in G$ *there are* $C, D \in G$ *such that* $E \sim lgg(C, D)$ *and* $d^{(1)}_{\perp}(E) = d^{(1)}_{\perp}(C) + d^{(1)}_{\perp}(D)$ *and* $|d^{(1)}_{\perp}(C) - d^{(1)}_{\perp}(D)| \leq 1$.

Theorem 4. *Let lattice* $\langle G, \succeq \rangle$ *be divisible with respect to* ρ. *Then for all* $C \in G \backslash \{\perp\}$ *we have* $d^{(2)}_{\perp}(C) \leq \lceil log_2(d^{(1)}_{\perp}(C)) \rceil + 1$.

Proof. The proof is similar to the proof of this theorem for general downward binary refinement operators (Theorem 16 in [5]). □

Theorem 5. *Lattice* $\langle \overrightarrow{\mathcal{L}}_{\perp}, \succeq_{\perp} \rangle$ *is divisible with respect to* $\rho_{\perp}$ *where* $\rho_{\perp}$ *is an upward refinement operator for this lattice.*

Sketch proof. We need to show that the conditions in Definition 12 hold for refinement operator $\rho_{\perp}$. The proof follows from previous results on refinement operators relative to $\perp$ that there are finite chains of upward covers from $\overrightarrow{\perp}$ to $\overrightarrow{C}$, $\overrightarrow{D}$ and $\overrightarrow{E}$ and from $\overrightarrow{E}$ to $\overrightarrow{C}$, $\overrightarrow{D}$ (Lemma 6 and Lemma 7 in [16]) and that the refinement steps from $\overrightarrow{\perp}$ to $\overrightarrow{E} = lgg_{\perp}(\overrightarrow{C}, \overrightarrow{D})$ is defined from the refinement steps from $\overrightarrow{\perp}$ to $\overrightarrow{C}$ and $\overrightarrow{D}$ (Proposition 5 in [16]). □

From Theorem 5 and Theorem 4 it follows that the minimum length of σ-chains for upward stochastic binary refinement operator relative to $\perp$ is logarithmically related to the minimum length of σ-chains for upward stochastic unary refinement operator relative to $\perp$.

Proposition 2. *Let* $S_0 \xrightarrow{\sigma_1} S_1 \xrightarrow{\sigma_1} S_2 \xrightarrow{\sigma_1} \ldots$ *and* $S_0 \xrightarrow{\sigma_2} S'_1 \xrightarrow{\sigma_2} S'_2 \xrightarrow{\sigma_2} \ldots$ *be stochastic refinement searches such that* σ_1 *is an upward unary and* σ_2 *an upward binary stochastic refinement operators relative to* $\perp$. *Then the minimum length of* σ_2*-chains from* S_0 *to a target clause is logarithmically related to the minimum length of* σ_1*-chains from* S_0 *to the target clause.*

Proposition 2 suggests that a stochastic refinement search which uses a binary refinement operator relative to $\perp$ (e.g. $lgg_{\perp}$, $armg_{\perp}$) requires logarithmically less refinement steps to find a target clause compared to a stochastic refinement search which uses a unary refinement operator. In other words, a binary stochastic refinement has a logarithmic convergence relative to a unary stochastic refinement.

The ILP system ProGolem [6] uses Asymmetric Relative Minimal Generalisation (ARMG or $armg_{\perp}$), which as mentioned in Section 4.3, can be viewed as stochastic binary refinement operator relative to $\perp$. ARMGs are constructed by adding randomly sampled positive examples at each iteration. According to Theorem 3, when using *lgg* with a proper sample size one could guarantee that with a high probability a consistent clause is generated. This theorem is also true for *ARMG* and it can be shown that with a high probability a consistent *ARMG* can be generated. According to Proposition 1, we have a monotonic increase in the positive coverage and Proposition 2 suggest that the stochastic binary refinement search used in ProGolem should have a logarithmic convergence. The logarithmic convergence described in Proposition 2 is consistent with quick convergence of ProGolem on different datasets (e.g. Fig. 5.b in [6]).

6 Conclusions

The refinement graph theory has been viewed as the main theoretical foundation of ILP [9]. Since the publication of this theory, there have been attempts to build ILP systems based on stochastic and randomised methods. However, to date there are very little theory to support the developments of these systems. We believe this paper is a first step in this direction. In this paper we discussed how the refinement theory and relevant concepts such as refinement operators can be adapted for a stochastic ILP search. Stochastic refinement is introduced as a probability distribution over a set of clauses which can be viewed as a prior in a stochastic ILP search. We gave an analysis of stochastic refinement operators within the framework of stochastic refinement search. We studied the properties of a stochastic refinement search as two well known Markovian approaches: 1) a Gibbs sampling algorithm and 2) a random heuristic search. We have shown that a minimum sample size can be set so that in each iteration a consistent clause is generated with a high probability. A stochastic refinement search can be viewed as a monotonic random heuristic search. The advantage of studying stochastic refinement search as a random heuristic search is that we can use the theoretical results from random heuristic search in order to analyse the behaviour and convergence of the search. This paper also provided some theoretical insights to explain the relative advantages of using stochastic lgg-like operators as in the ILP systems Golem and ProGolem. The proposed framework in this paper can be extended in several ways. For example, it is possible to define stochastic refinement operators and stochastic refinement search for theories. This is especially important as the search space of theories are more complex and more difficult to search and randomised and stochastic methods should work better. As another future work, we intend to explore methods for learning probabilities in stochastic refinement operators. This could be similar to learning probabilities for a Stochastic Logic Program (SLP) [7], especially that stochastic refinement operators can be easily implemented as SLPs.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments and A. Srinivasan for helpful discussions. The first author was supported by the BBSRC grant BB/C519670/1. The second author thanks the Royal Academy of Engineering and Microsoft for funding his present 5 year Research Chair.

References

1. Gelman, A., Carlin, J., Stern, H., Rubin, D.: Bayesian Data Analysis. Chapman and Hall/CRC, Boca Raton (2003)
2. Haussler, D., Kearns, M., Shapire, R.: Bounds on the sample complexity of Bayesian learning using information theory and the VC dimension. Machine Learning 14(1), 83–113 (1994)
3. Mitchell, T.: Machine Learning. McGraw-Hill, New York (1997)
4. Muggleton, S., Feng, C.: Efficient induction of logic programs. In: Muggleton, S. (ed.) Inductive Logic Programming, pp. 281–298. Academic Press, London (1992)
5. Muggleton, S., Marginean, F.: Binary refinement. In: Proceedings of Workshop on Logic-Based Artificial Intelligence (1999)
6. Muggleton, S., Santos, J., Tamaddoni-Nezhad, A.: ProGolem: a system based on relative minimal generalisation. In: De Raedt, L. (ed.) ILP 2009. LNCS, vol. 5989, pp. 131–148. Springer, Heidelberg (2010)
7. Muggleton, S.H.: Stochastic logic programs. In: De Raedt, L. (ed.) Advances in Inductive Logic Programming, pp. 254–264. IOS Press, Amsterdam (1996)
8. Muggleton, S.H., Tamaddoni-Nezhad, A.: QG/GA: A stochastic search for Progol. Machine Learning 70(2-3), 123–133 (2007)
9. Nienhuys-Cheng, S.-H., De Wolf, R.: Foundations of Inductive Logic Programming. LNCS (LNAI), vol. 1228. Springer, Heidelberg (1997)
10. Paes, A., Zelezny, F., Zaverucha, G., Page, D., Srinivasan, A.: ILP Through Propositionalization and Stochastic k-Term DNF Learning. In: Muggleton, S.H., Otero, R., Tamaddoni-Nezhad, A. (eds.) ILP 2006. LNCS (LNAI), vol. 4455, pp. 379–393. Springer, Heidelberg (2007)
11. Pompe, U., Kovacic, M., Kononenko, I.: SFOIL: Stochastic approach to inductive logic programming. In: Proceedings of the Second Electrotechnical and Computer Science Conference ERK, vol. 93, pp. 189–192. Citeseer (1993)
12. Ruckert, U., Kramer, S.: Stochastic Local Search in k-term DNF Learning. In: Proc. 20th International Conf. on Machine Learning, pp. 648–655 (2003)
13. Serrurier, M., Prade, H., Richard, G.: A simulated annealing framework for ILP. In: Camacho, R., King, R., Srinivasan, A. (eds.) ILP 2004. LNCS (LNAI), vol. 3194, pp. 288–304. Springer, Heidelberg (2004)
14. Srinivasan, A.: The Aleph Manual. University of Oxford, Oxford (2007)
15. Tamaddoni-Nezhad, A., Muggleton, S.H.: A genetic algorithms approach to ILP. In: Matwin, S., Sammut, C. (eds.) ILP 2002. LNCS (LNAI), vol. 2583, pp. 285–300. Springer, Heidelberg (2003)
16. Tamaddoni-Nezhad, A., Muggleton, S.H.: The lattice structure and refinement operators for the hypothesis space bounded by a bottom clause. Machine Learning 76(1), 37–72 (2009)
17. Vose, M.D.: Random heuristic search. Theoretical Computer Science 229(1), 103–142 (1999)
18. Zelezny, F., Srinivasan, A., Page, D.: Randomised restarted search in ILP. Machine Learning, 64 1(3), 183–208 (2006)

Fire! Firing Inductive Rules from Economic Geography for Fire Risk Detection

David Vaz[1], Vítor Santos Costa[2], and Michel Ferreira[3]

[1] LIACC and DCC/FCUP, University of Porto, Portugal
[2] CRACS-INESC Porto LA and DCC/FCUP, University of Porto, Portugal
[3] IT and DCC/FCUP, University of Porto, Portugal

Abstract. Wildfires can importantly affect the ecology and economy of large regions of the world. Effective prevention techniques are fundamental to mitigate their consequences. The design of such preemptive methods requires a deep understanding of the factors that increase the risk of fire, particularly when we can intervene on these factors. This is the case for the maintenance of ecological balances in the landscape that minimize the occurrence of wildfires. We use an inductive logic programming approach over detailed spatial datasets: one describing the landscape mosaic and characterizing it in terms of its use; and another describing polygonal areas where wildfires took place over several years. Our inductive process operates over a logic term representation of vectorial geographic data and uses spatial predicates to explore the search space, leveraging the framework of Spatial-Yap, its multi-dimensional indexing and tabling extensions. We show that the coupling of a logic-based spatial database with an inductive logic programming engine provides an elegant and powerful approach to spatial data mining.

1 Introduction

Wildfires are an unavoidable event in Nature and play an important role in wildland ecosystems. Naturally caused wildfires are, however, a small percentage of all the wildland fires. Preventing and mitigating the consequences of wildfires that result from the increased pressure of human activity in wildland areas has been the goal of fire control programs for more than a century. Prevention techniques range from measures aiming to reduce human infractions, to the altering of stored fuels, through controlled burns, in wildlands to affect future fire risk and behavior. In addition to the straightforward impact of fuels and weather conditions in the occurrence of fires, the role of topography is also relevant. Here we understand topography in a broader sense, as a discipline concerned with local detail of space, including not only relief but also vegetation and human-made features. In areas where human intervention has importantly reshaped this topography, as happens in many European regions where native forests have been replaced by fast-growing trees and pasture areas, the impact of this human-designed organization of landscape can potentially affect the occurrence and behavior of wildfires. In this paper we focus on this landscape organization factor, which has been given little attention by fire control programs. While some aspects of the human-made organization of landscape can obviously affect fire behavior, such as the existence of major roads cutting through forest areas, that act as barriers to the propagation of fire, there are also potentially less obvious correlations between this local organization of the landscape and the occurrence of fires, which can profit from machine learning techniques. Understanding such correlations can then guide the human intervention in the landscape towards more efficient prevention and mitigation of the consequences of wildfires.

In this paper we propose the use of inductive logic programming (ILP) to design logic theories that correlate the local organization of the landscape with the occurrence of fires, using the framework of Spatial-Yap [1] to have term-based representation of vectorial geographic data. In addition, Spatial-Yap provides a logic-based approach to a geographic information system and is able to render the inductive engine with spatial relationship predicates that are used to formulate theories *on-the-fly* based on spatial reasoning. We use detailed spatial databases that contain vectorial representations of the landscape organization in the north of Portugal. We overlay these databases with another spatial database that keeps historical records of wildfires taking place over several years in the same region.

The remainder of this paper is organized as follows. The next section presents related work on the use of ILP with spatial datasets. Section 3 describes our fire dataset and the methodology we used. In Section 4 we report preliminary results, together with a discussion of these results. Finally, Section 5 ends the paper.

2 Related Work

Over the past years, the use of spatial data has increased in many areas of computer science. Relational Database Management Systems (RDBMS) were among the first systems to tackle this kind of data, both through extensions to support spatial data, and by providing functions to manipulate the data.

The Open Geospatial Consortium (OGC) proposed a standard to extend SQL-92 in “OpenGis Simple Features Specification for SQL” (OGC99) [2]. The purpose of this specification is to define a standard SQL schema that supports storage, retrieval, query and update of simple geospatial feature collections. Examples of RDBMS systems that conform to this standard are Oracle Spatial and PostgreSQL, through the PostGIS module.

The development of such sophisticated geographical databases has led to interest in *spatial data mining*, defined to be the branch of data-mining where the spatial neighbors of an object may have an influence on the object [3]. A typical task would be to find clusters of correlated objects [4], but a large number of different applications are possible.

Arguably, spatial learning can be considered as an instance of multi-relational learning, with a very specific type of domain knowledge, and should be an important application for ILP. Malerba [3] and his group have exploited this approach with very interesting results. In their approach, multi-relational data-mining techniques are applied by working at a higher conceptual level of the geographic information [5]. Their approach follows a two step algorithm. First, system such as INGENS [6] extract relevant concepts and features from a spatial database, by applying and expanding on standard GIS tools. Second, this relational representation of spatial data can be mined by ILP techniques: ATRE [7] implements a sequence coverage algorithm that learns a classifier, and SPADA [8] is an association-rule learner that can find strong spatial association rules.

The INGENS work raises a number of interesting questions. One important problem, discussed by Malerba [5], is the computational cost of performing feature extraction: although spatial facts are rarely updated, attribute expansion can be expensive in terms of time and space, with often time being spent computing unnecessary attributes. One would expect this problem to grow as databases grow in size and complexity.

One possible approach to this problem is to couple a database to a deductive system: MYDDAS [9] couples YapTab [10] and MySQL extended with geometry types to form Spatial-Yap [1] (unfortunately MySQL has never evolved to conform with OGC99). In this paper we take the next step and actually couple tightly Prolog inference with the geographical data itself. In order to perform inference with logic programming, we need to address well the three key components of geographical data-mining:

1. Spatial terms for representing and storing spatial objects.
2. Spatial predicates, to manipulate (e.g., intersect two spatial terms) and to query spatial terms toward finding interesting properties such as area or distance between two spatial terms.
3. Effective indexing of spatial terms, not only because of the usual mammoth size of such terms, but also because of the number of different terms in the database and the complexity of spatial predicates.

We address the first problem by simply using Prolog terms to represent the three main geometry types, as they are presented in OGC99 standard: *Point*, *Linestring* and *Polygon*; and collection types. We support the OGC defined attributes and restrictions, as can be consulted in the OGC99 document [2]. Our representation is thus similar to the Well Known Text of OGC99, with spatial terms conforming with the grammar in Figure 1.

We further define a set of spatial predicates that provide an interface to the GEOS API [11], that conforms to the OGC99 standard and is also used by PostGIS. Table 1 summarizes these predicates.

This machinery provides the foundation for a logic programming geographical information system. The next step was based on the observation that spatial data does not benefit from most of the traditional indexing techniques (namely the ones used in the logic programming), as most of them are based on single dimension indexing structures.The RDBMS community addressed this problem by proposing novel data-structures, namely `R`-Trees which have become standard [12].

We extended Prolog indexing through *User Defined Indexing* (UDI) [13], a new extension to Prolog indexing where the programmer is able to define the indexing mechanism based on *what* the terms in the arguments of a predicate

```
SpatialTerm = Point | LineString | Polygon
            | MultiPoint | MultiLineString | MultiPolygon | GeometryCollection ;
Point = "point" PointTerm ;
LineString = "linestring(" PointTermList ")" ;
Polygon = "polygon(" PointTermListList ")" ;
MultiPoint = "multipoint(" PointTermList ")" ;
MultiLinestring = "multilinestring(" PointTermListList ")" ;
MultiPolygon = "multipolygon(" PointTermListListList ")" ;
GeometryCollection = "geometrycollection(" SpatialTermList ")" ;
PointTerm = "(" Number "," Number ")" ;
```

Fig. 1. EBNF of Spatial Terms

Table 1. Spatial Predicates

Type	**Predicate**	

Predicates for testing spatial properties	ogc_is_empty(+Geom)
	ogc_is_simple(+Geom)	

	ogc_equals(+Geom1,+Geom2)
	ogc_disjoint(+Geom1,+Geom2)
	ogc_touches(+Geom1,+Geom2)
	ogc_within(+Geom1,+Geom2)
	ogc_overlaps(+Geom1,+Geom2)
	ogc_crosses(+Geom1,+Geom2)
	ogc_intersects(+Geom1,+Geom2)
	ogc_contains(+Geom1,+Geom2)
	ogc_relate(+Geom1,+Geom2,?PatternMatrix)	

Predicates that support spatial analysis	ogc_envelope(+Geom,?GeomEnvelope)
	ogc_boundary(+Geom,?GeomBoundary)
	ogc_buffer(+Geom,+Distance,?GeomBuffer)
	ogc_convex_hull(+Geom,?GeomConvexHull)
	ogc_intersection(+Geom1,+Geom2,?GeomIntersection)
	ogc_union(+Geom1,+Geom,?GeomUnion)
	ogc_difference(+Geom1,+Geom,?GeomDifference)
	ogc_symmetric_difference(+Geom1,+Geom2,?GeomSymDiff)
	ogc_distance(+Geom1,+Geom2,?Distance)	

Specific type predicates	Linestring
	Multilinestring
		ogc_length(+Geom,?Length)
		ogc_is_closed(+Geom)
		ogc_is_ring(+Geom)

	Polygon
	Multipolygon
		ogc_area(+Geom,?Area)
		ogc_centroid(+Geom,?GeomPoint)
		ogc_point_on_surface(+Geom,?GeomPoint)

are meant to represent. This allows users to provide an indexing function that selects a subset of the clauses of a predicate, given a set of constrained variables or bound Prolog terms.

In this work, we use spatial terms [1]. As discussed above, these are simple geometry types based on 2D points. Notice that the simplicity of the primitives does not mean that the terms themselves are simple. For example, the polygons shown in the figures of this paper are represented in Prolog with several hundred points each.

The key idea in R-Trees is to use the Minimum Bounding Rectangle (MBR) to index data. Each leaf nodes stores an object (or at least a pointer), and is keyed by the object's MBR. Inner nodes are keyed by an MBR that is the union of all MBRs below. Notice, that in contrast to single dimension indexing, keys cannot be sorted as there is no order. Nevertheless, on most datasets the tree will maintain a form that allows the search algorithm to quickly discard irrelevant regions.

Figure 2 shows an example R-Tree designed to store the boundaries of European countries. Figure 2(a) details part of the index structure, and Figure 2(b) graphically depicts the actual boundaries and MBRs that define the R-Tree. Notice that although European countries do not overlap, their MBRs do. The tree has height 3. The root node (Level 3) contains two MBRs, R_1 and R_2, shown as the wider (blue) lines. Notice that there is some overlap, as we cannot find a disjoint balanced union of MBRs that covers the whole of Europe. The overlap is even more evident on Level 2, Also observe that whereas Iceland, Greece and Portugal belong to a single box *for each level*, the central Alps region in Europe is covered by a large number of overlaping MBRs *at all levels.*

The main query we use in this application is the `overlaps` binary constraint, `&&`, also the key operator on the Postgis spatial RDBMS [14]: `A && B` *constraint is satisfied if* `A`*'s bounding box overlaps* `B`*'s bounding box.*

A query using this operator is shown next:

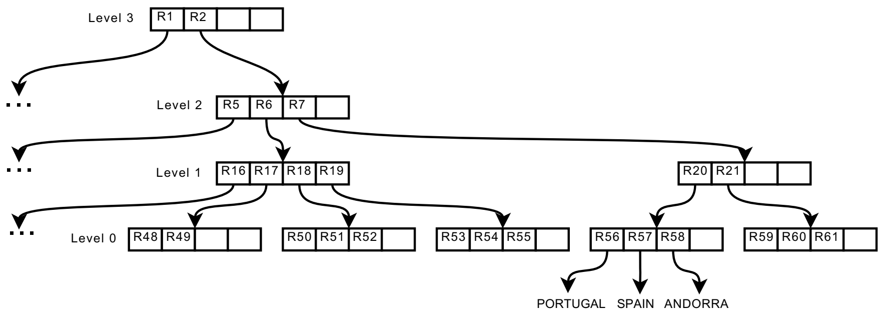
```
?- country(spain,P1), P2 && P1, country(Country,P2).
```

The first sub-goal instantiates `P1` to the polygon for `spain`. Thus, `P2 && P1` are called with `P1` bound and `P2` will be attributed a value by `overlap(_,P1)`. The benefit is that the second call to `country` will only search the database for countries that have overlapping boundaries with *spain.*

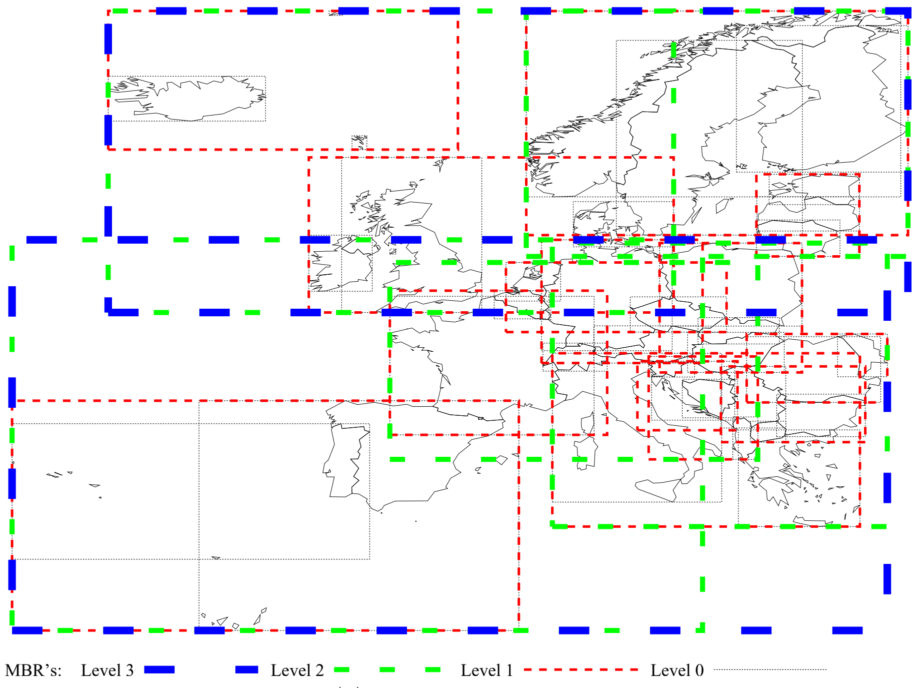
We should remark that `&&` only approximates overlapping, based on MBRs. In our implementation we perform intersection explicitely, although intersection could naturally be performed within the constraint solver. For example:

```
?- country(spain,P1), P2 && P1, country(C,P2),
   intersection(P1,P2,P3).
```

The query searches for countries that intersect with Spain. The overlapping constraint prunes the results to Portugal, France and Andorra, but only the latter will eventually succeed. Notice that the same result would be achieved without the use of UDI, but with a high penalization in time:



(a) R-Tree Structure



(b) MBR Containment

Fig. 2. R-Tree of Europe Countries

```
?- country(spain,P1), country(C,P2), overlap(P1,P2),
   intersection(P1,P2,P3).
```

Our results show that using this form of indexing is fundamental in operating effectively with spatial data.

3 The Fire DataSet

Portugal being the smallest of the five southern Europe countries, is the most affected by fire in terms of occurrences and relative burnt area. From 1980 to 2004, 30% of the country was burnt (equivalent to 1 fire per 20ha). The closest cases (Italy and Spain) present values of fire occurrence, density and burnt area inferior by 1/3 and 1/5 respectively.

Given the increasing trend of burnt area and with the increasing changes in temperature and precipitation, it is of the outmost importance to narrow the problematic areas in order to effectively promote fire control and landscape management. Recent efforts have provided detailed information of burnt areas between 1990 and 2007.

Initial work on this area has focused on a parish-based approach, where the goal has been to study differences between different regions in the country [15], In related work, Stojanova et al use geographic and weather data to predict forest fires [16]. A number of different classifiers and regression techniques were applied, with best results obtained through bagging decision trees.

The motivation for our work was the need to consider different sources of data, given that we have both parish data and the COS'90 database, a detailed land cover map, produced by the National Center for Geographic Information. The COS'90 database was obtained by visual interpretation of aerial photographs from 1990 followed by polygon vectorization, with 3 digit nomenclature for each polygon. The nomenclature describes the principal and secondary type of use, e.g., `PE2` would express a polygon with a mixed forest based on Pine Tree and Eucalyptus covering up to 75% of the area. The order of the first letters informs that Pine Tree is dominant, the digit informs that we have between 30% to 50% of coverage.

We further remark that polygons vary widely in size. Moreover, polygons do overlap with each other, and we have cases of polygons that are contained in other polygons. We do not exploit such overlaps in this work.

Given the fine grained level of this dataset and the absence of detail in burnt areas polygons, e.g., a given burnt area polygon may represent several fires occurring with a spawn of several months within a year, we have abstracted the burnt area by tagging the COS'90 polygons with a `burnt` label for each year, making this layer our base layer. We have only used burnt information from 1991 to 1999, an acceptable spawn given the base date of COS'90.

Additional data information can be obtained by considering a second layer with socioeconomic information. In Portugal this data is available through statistics taken over parishes.

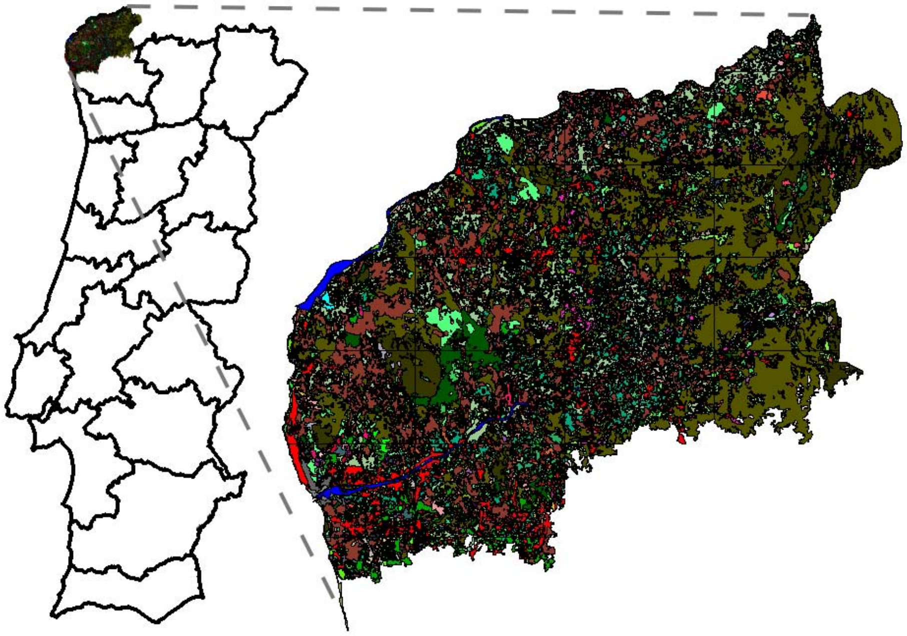


Fig. 3. COS'90 Polygons in Viana do Castelo

We focus on the Viana do Castelo district (county) - see Figure 3. This district is one of the most heavily forested in Portugal, and has suffered from a wide diversity of fires. Previous work indicates that different regions have very different patterns: Viana do Castelo is typical of the North of Portugal and is one of the most affected sub-regions of the country. The district has 290 parishes and 15091 COS'90 polygons.

Notice that the relation between parishes and polygons can be quite complex. Figure 4 shows a situation where a large polygon overlaps a number of parishes. The opposite is also possible, and a small polygon can be easily contained in a parish.

3.1 Methodology

In this work we are interested in exploring spatial predicates on-the-fly during the ILP search process. We thus rely on spatial indexing to obtain more efficient execution of queries involving spatial predicates. Notice that even with spatial indexing, geographical queries are typically very expensive. We further use *tabling* to reduce recomputation to the minimum. As an example of this type of optimization, consider the *neighbor* relation:

```
:- table neighbor/2.
neighbor(ID1,ID2) :-
```

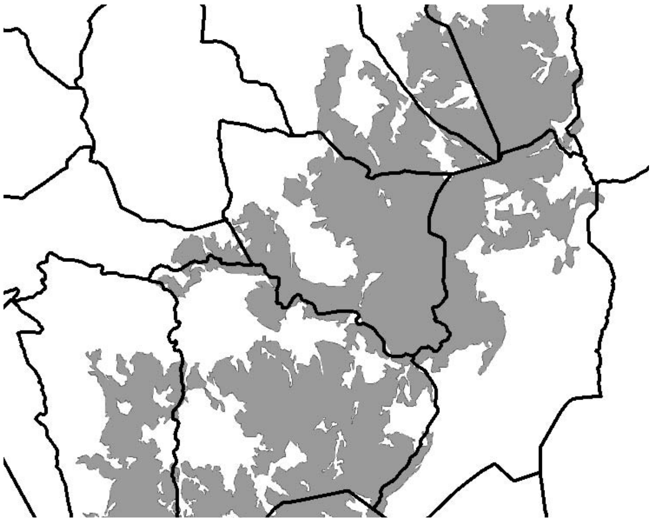


Fig. 4. Polygons and Parishes. Notice the irregular structure of the polygon and how it crosses over a number of parishes.

```
cos90(ID1,R1,P1), R2&&R1, cos90(ID2,R2,P2), ogc_touches(P1,P2).
```

Logically, it would be sufficient to express the `cos90` polygons and then use `ogc_touches`. Assuming that R_1 is bound, the `&&` constraint selects polygons such that R_2 overlaps R_1, R_1 and R_2 are the Minimum Bounding Rectangle (MBR) for the polygons P_1 and P_2 respectively. This is a necessary but not sufficient condition for the actual polygons to touch. The relation `ogc_touches` holds true if and only if the two polygons touch.

Notice that `&&` does not introduce any new logical information. On the other hand, `&&` is implemented very effectively by using UDI with `R`-Trees. In contrast, `ogc_touches` is extremely expensive: it needs to compare two complex polygons. Our approach reduces very significantly the number of calls to `ogc_touches` and makes the whole computation feasible.

However, it is clear that the `neighbor` operation will be used quite often, and may have to be recomputed every time we run a rule. We use tabling to avoid this problem. More precisely, we use tabling with *local* scheduling so that *all* solutions to the query are computed the first time the query is run.

A second interesting problem arises from the need to join the two base layers: how do parishes match COS'90? Both cover the same area, but they have different granularity and they have different information associated with it. They even overlap each other as seen in the example in Figure 4. In general, we do not expect to find an ideal solution to this problem: but in order to use this data we have used a weighted average based on the area of the intersection of both layers.

```
expbox(ID1,Class,ID2,Distance) :-
  expbox(ID1,Class,0,ID2,Distance), !.

expbox(ID1,CL,0,ID2,DISTANCE) :-
  cos90(ID1,R1,P1), R2&&R1,
  findall((ID2,D),
          (cos90(ID2,R2,P2),class(ID2,CL),ogc_distance(P1,P2,D)), L),
  mindistance(L,ID2,DISTANCE).
expbox(ID1,CL,Expand,ID2,Distance) :- Expand > 0,
  cos90(ID1,R1,P1), expand(R1,Expand,R1E), R2&&R1E,
  findall((ID2,D),
          (cos90(ID2,R2,P2),class(ID2,CL),ogc_distance(P1,P2,D)), L),
  mindistance(L,ID2,Distance).
expbox(ID1,CL,Expand_,ID2,Distance) :-
  Expand is Expand_ + 10000, expbox(ID1,CL,Expand,ID2,Distance).
```

Fig. 5. Neighbor Search in the Background Knowledge

Spatial distance between two spatial objects corresponds to the minimum distance between any two points of each spatial object. Hence to find the minimum distance to a water class polygon, for example, we would need to calculate the distance to each water class polygon. In this case, indexing is not straightforward, but is still worthwhile. The R-Tree indexing structure abstracts spatial objects to its Minimum Bounding Rectangle, and is stored in a form that allows us to discard spatial objects far from the search rectangle. Using the indexing structure we can speedup minimum distance calculations by expanding gradually the search rectangle, starting from the MBR of the base polygon, until a matching polygon is found. A version of the algorithm is presented in Figure 5.

3.2 The Background Knowledge

We can now present the background knowledge used in this experiment. We combine a number of different information sources.

The `class/2` relation identifies all the activities pertaining to the polygon. It is defined in Prolog as:

```
class(ID,C) :-
      pol(ID, CL),
      sub_atom(CL,_,_,_,C), C \= ''.
```

The area relation corresponds to the `ogc_area/2` relation discussed above, and gives the polygon's area. The `neighbor/2` relation corresponds to `ogc_touches` and gives the connection between different polygon.

As an example of temporal correlated information, we also have relations saying whether a polygon was burnt in the previous year or whether it was burnt ever.

Besides `class` and neighbor information, we use information from parishes, currently the amount of cattle on a certain parish. As a polygon may intersect several different parishes, we estimate the cattle in a polygon using a weighted average of cattle based on the intersection area.

Last, we use the `geq/2` and `leq/2` relations to handle numeric data.

4 Results and Discussion

Our task is to predict whether a polygon will catch fire. We recall that forest fires are complex events with a large variety of causes. We would not expect to be able to predict exactly which polygons will take fire. On the other hand, it is worthwhile to find rules that are highly indicative of vulnerability to fire.

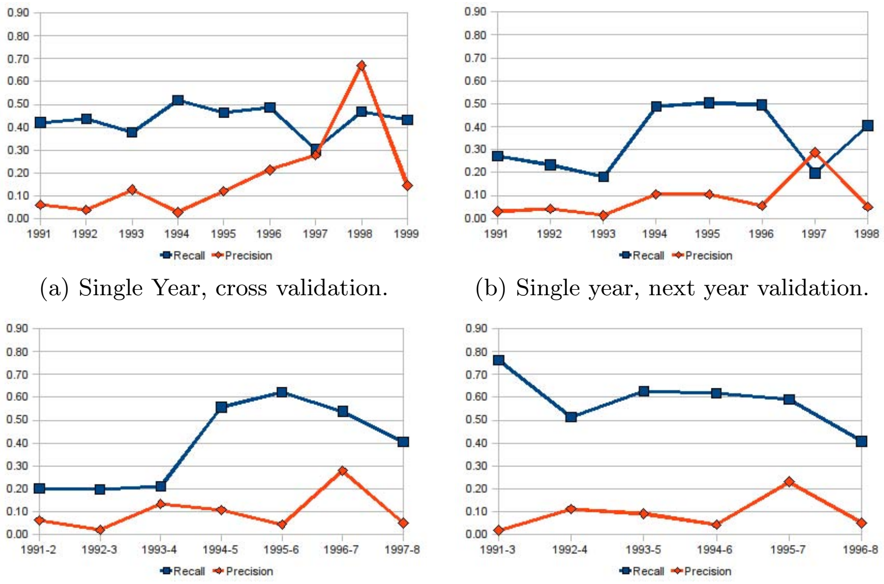
We use the ILP system Aleph [17] running under the Prolog system YAP-6 [18] to search for fire risk areas. As discussed above we performed this study on the years from 1991 to 1999. In each year, positive examples (COS'90 polygons with fire occurrence) range from 180 to 1834 occurrences. We used as negative examples the remainder of 15091 polygons in the dataset. The dataset is therefore highly skewed.

We follow two different types of approaches: first, we use cross-validation over the different years; second, we try to predict the *next* year. In the latter case, we can learn with multiple years: we use up to three consecutive years. To evaluate runs on the same year we used stratified 10-fold cross validation. Results can be seen in Figure 6.

Figure 6(a) shows system performance at every year. Given that the dataset is very skewed, we use precision and recall as a measure of performance. Recall performance on the test set tends to range around 50%, and precision around 10%. We find these values quite acceptable, given the nature of the problem and the skew of the dataset (only about 2% of the examples are positives). Notice that the results vary significantly according to each year. In general, precision tends to be best for the years with most fires. In contrast, recall tends to be worse for these years: this is because we learn less rules in these years. The results for 1998 are quite interesting. This year about 1800 polygons burned (10% of COS'90), and the following single rule is highly predictive:

```
burnt(A,E) :-
   burnt_before(A,E).
```

Figure 6(b), 6(c), and 6(d) show next year validation performance with one year, two year, and three year training. Because years are widely different, testing the rules on the next year tends to have poorer performance than using the same year. On the other hand, as we use more years to train the system, recall and precision improve and approach same year training. Moreover, performance becomes more stable and less sensitive to variations in a year (on the other hand, we cannot take advantage of special years such as 1998). In general, with 3 year training we get a recall over 60%, with a precision of about 10%.



(a) Single Year, cross validation.

(b) Single year, next year validation.

(c) Two years, next year validation.

(d) Three years, next year validation.

Fig. 6. Results

Two examples that give a flavor of the rules learned by our system:

```
burnt(A,_) :-
   class(A,'II'), water_pol(A,_,B), B >= 6736.85994035496.
burnt(A,E) :-
   parish(A,B), sheep(B,_,_,C), C >= 34,
   neighbor(A,D), burnt_last_year(D,E), class(D,'II').
```

The first rule refers to a polygon classified as "improductive" i.e. fallow land. The rule states that such land is quite likely to burn if more than 6Km away from a water source. The second rule applies to a polygon that is in a rural area with a high increase in sheep population, and close to fallow land that often burns. Rules also have a geographic interpretation. Figure 7 graphically shows coverage for the second rule. Notice that most polygons covered by the rule are in the interior, more precisely, on the mountain regions of Viana. Notice also that the rule mostly refers to contiguous regions. In general the found rules most often refer to previous fire activity, to types of vegetative cover such as fallow lands, brush, pine and oak, to herding and to distance to water. Also, a high percentage of rules refer to neighboring polygons.

Exporting the Rules As a way of evaluating the usefulness of our rules, we experimented with applying rules learned in Viana do Castelo on a different district (county). We experimented with Braga district, the southern neighboring district to Viana. Braga shares many of the traits in Viana, but is a larger and

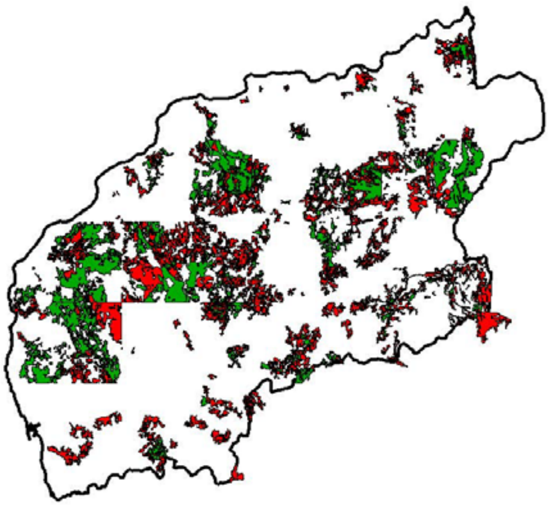
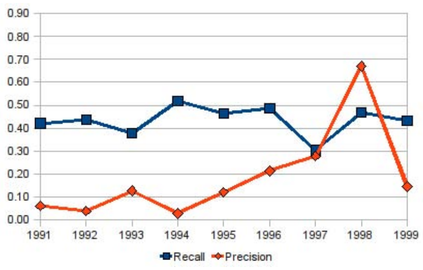
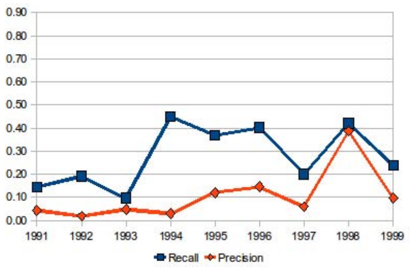


Fig. 7. Positive and Negative Coverage of Rule2 in Viana do Castelo



(a) Single Year, next year validation in Viana do Castelo.



(b) Single Year, next year validation in Braga.

Fig. 8. Comparing Results in Braga and Viana do Castelo

more complex region. It is also more heavily populated, with a smaller portion of forest area. Otherwise, the types of occupation are similar in both regions.

Figure 8 compares results for rules learned in one year and applied to the next year in Viana and Braga. Notice that there is a strong correlation between the two curses. On the other hand, the results are somewhat worst for Braga than for Viana, as expected, but still better than default accuracy.

5 Conclusions

In this paper we have presented an ILP approach to spatial data mining, addressing the pressing problem of wildfire prevention through the understanding of the impact of landscape organization. Our work leverages the machinery we developed in previous research, namely in the construction of an OGC-compliant logic-based geographic information system. A fundamental contribution of this work results from the coupling of an ILP system with a logic-based geographic information system. This coupling avoids the off-line materialization step of spatial features using external geographic information systems, allowing the search process to dynamically explore spatial relationship predicates in the formulation of clauses. The use of multi-dimensional indexing and tabling prove to be also crucial for the computational feasibility of our approach, providing an additional contribution for the use of ILP in the context of spatial data mining with real-world datasets.

Acknowledgments

This work has been partially supported by projects JEDI (PTDC/EIA/66924/2006), STAMPA (PTDC/EIA/67738/2006), and HORUS (PTDC/EIA/100897/2008), and by funds granted to LIACC, CRACS and IT through the Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia and Programa POSC. David Vaz is funded by FCT PhD grant SFRH/BD/29648/2006.

We also thank João Honrado and João Torres for providing the fire dataset and for the preliminary discussions in understanding the role of landscape topography in the occurrence of fires.

References

1. Vaz, D., Ferreira, M., Lopes, R.: Spatial-yap: A logic-based geographic information system. In: Dahl, V., Niemelä, I. (eds.) ICLP 2007. LNCS, vol. 4670, pp. 195–208. Springer, Heidelberg (2007)
2. Open GIS Consortium, I.: OpenGIS Simple Features Specifications For SQL (1999), http://www.opengis.org/docs/99-049.pdf
3. Ceci, M., Appice, A., Loglisci, C., Caruso, C., Fumarola, F., Malerba, D.: Novelty detection from evolving complex data streams with time windows. In: Rauch, J., Raś, Z.W., Berka, P., Elomaa, T. (eds.) ISMIS 2009. LNCS, vol. 5722, pp. 563–572. Springer, Heidelberg (2009)
4. Ng, R.T., Han, J.: Efficient and effective clustering methods for spatial data mining. In: Bocca, J.B., Jarke, M., Zaniolo, C. (eds.) VLDB 1994, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, pp. 144–155. Morgan Kaufmann, Santiago de Chile (1994)
5. Malerba, D., Lanza, A., Appice, A.: 10. In: Geographic Knowledge Discovery and Data Mining, 2nd edn., pp. 258–291. CRC Press - Taylor and Francis (2009)
6. Malerba, D., Esposito, F., Lanza, A., Lisi, F.A., Appice, A.: Empowering a gis with inductive learning capabilities: the case of ingens. Computers, Environment and Urban Systems 27(3), 265–281 (2003)
7. Malerba, D.: Learning recursive theories in the normal ilp setting. Fundam. Inf. 57(1), 39–77 (2003)
8. Lisi, F.A., Malerba, D.: Inducing multi-level association rules from multiple relations. Mach. Learn. 55(2), 175–210 (2004)

9. Soares, T., Ferreira, M., Rocha, R.: The MYDDAS Programmer's Manual. Technical Report DCC-2005-10, Department of Computer Science, University of Porto (2005)
10. Rocha, R., Silva, F., Santos Costa, V.: YapTab: A Tabling Engine Designed to Support Parallelism. In: Conference on Tabulation in Parsing and Deduction, pp. 77–87 (2000)
11. The GEOS Development Team: GEOS: Geometry Engine Open Source, `http://geos.refractions.net/`
12. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Yormark, B. (ed.) SIGMOD 1984, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, pp. 47–57. ACM Press, New York (1984)
13. Vaz, D., Santos Costa, V., Ferreira, M.: User defined indexing. In: Hill, P.M., Warren, D.S. (eds.) ICLP 2009. LNCS, vol. 5649, pp. 372–386. Springer, Heidelberg (2009)
14. The Postgis Development Team: Postgis adds support for geographic objects to the postgresql object-relational database, `http://postgis.refractions.net/`
15. Torres, J., GonÃ∮alves, J., Torgo, L., Honrado, J.: Fire and landscape: A multiscale assessment of a complex realation. In: Landscape Ecology International Conference (2010)
16. Stojanova, D., Panov, P., Kobler, A., Džeroski, S., Taškova, K.: Learning to predict forest fires with different data mining techniques. In: Proceedings of the 9th International Multiconference Information Society 2006 (IS 2006), Jožef Stefan Institute, pp. 255–258 (2006)
17. Srinivasan, A.: The Aleph Manual (2001)
18. Santos Costa, V.: The life of a logic programming system. In: de la Banda, M.G., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 1–6. Springer, Heidelberg (2008)

Automating the ILP Setup Task: Converting User Advice about Specific Examples into General Background Knowledge

Trevor Walker[1], Ciaran O'Reilly[2], Gautam Kunapuli[1], Sriraam Natarajan[1], Richard Maclin[3], David Page[1], and Jude Shavlik[1]

[1] University of Wisconsin – Madison, USA
{twalker,kunapg,natarasr,shavlik,page}@biostat.wisc.edu
[2] SRI International
ciaran.oreilly@sri.com
[3] University of Minnesota, Duluth, USA
rmaclin@d.umn.edu

Abstract. Inductive Logic Programming (ILP) provides an effective method of learning logical theories given a set of positive examples, a set of negative examples, a corpus of background knowledge, and specification of a search space (e.g., via mode definitions) from which to compose the theories. While specifying positive and negative examples is relatively straightforward, composing effective background knowledge and search-space definition requires detailed understanding of many aspects of the ILP process and limits the usability of ILP. We introduce two techniques to automate the use of ILP for a non-ILP expert. These techniques include automatic generation of background knowledge from user-supplied information in the form of a simple relevance language, used to describe important aspects of specific training examples, and an iterative-deepening-style search process.

Keywords: Advice Taking, Human Teaching of Machines.

1 Introduction

Inductive Logic Programming (ILP) provides a method to learn logical theories that cover most of a given set of positive examples and as few as possible of a set of negative examples. Unlike many other supervised learning approaches, ILP often needs a complex corpus of background knowledge beyond just information provided as part of the example description (i.e., the example features). This information is vital to both forming the hypothesis space and guiding the search; effective use of ILP depends upon this background knowledge. The ILP-setup problem of articulating background knowledge can be difficult and requires detailed understanding of the ILP algorithm, greatly limiting ILP's usability by non-experts.

At least two possible solutions to this problem exist. One is a two-step process in which an ILP expert closely works with a domain expert in the first step to tailor the general-purpose ILP system to a specific domain, such as drug design (e.g., [3]). In the second step, domain experts, who are not ILP experts, can then use the tailored system. A second solution to this ILP-setup problem, which retains the general-purpose nature of the ILP system, is to allow a teacher to, as naturally as technically possibly, explain *why* specific examples are positive or negative through some advice language. This *teacher-provided advice* supplies hints about the concept being learned, beyond the traditional labeling of examples. Given this teacher-provided advice, the automated learner can generate background knowledge and set appropriate search parameters. This paper is the first study to explore this second approach, although some prior ILP work is related and is reviewed in Section 5.

Consider the following sample dialog between the teacher and the learner. Assume the formula $(p(X) \wedge q(X,Y)) \vee r(X)$[1] is a relevant piece of background knowledge for concept C. The teacher might express this indirectly via the following dialogue about a small number of training examples:

"In example 1, object a is a positive instance of concept C because $p(a)$ is true."

> Note that, in human instruction, the teacher might say this to mean simply that p is relevant to C rather than the complete definition of C.

"In example 2, object b is a positive instance of C because $r(b)$ is true."

> Note here that an algorithm that induces background knowledge from these statements needs to map both objects a and b to the same variable.

"In example 3, object d is a negative instance of C because $q(d, d)$ is false."

> Note that the teacher is telling the learner about relevant background knowledge through a negative example. The piece of advice in this case needs to be negated. In addition, the machine learner does not know whether the advice is about (1) all possible choices for the second (or first) argument of q, (2) restricting the choice of the second argument to be the same as the first, or (3) just the specific choice of constant d as the second (or first) argument.

Although $(p(X) \wedge q(X, Y)) \vee r(X)$ may be the formula necessary to define the concept, formulas such as $(p(X) \vee q(X, Y)) \wedge r(X)$, or $p(X) \wedge q(Y, X) \wedge r(X)$, or $p(X) \wedge (q(X, d) \vee r(X))$, or yet still others are also consistent with this human-provided advice.

Allowing the teacher to provide such advice permits the use of ILP in an unexplored setting in which only a few examples, along with teacher-provided annotations, are sufficient to learn the target concept. However, in a setting with few examples, while the target concept might be complex, such as $(p(X) \wedge q(X, Y)) \vee r(X)$, a simple clause, say $p(X)$, by itself might be sufficient to discriminate between examples. Thus, in this setting the advice should motivate the learner to prefer formulas that use all the teacher-mentioned predicates (i.e., p, q and r), rather than just the simplest formula consistent with the labeled examples.

Below, we present an algorithm to convert teacher-provided advice into ILP background knowledge. We designed this algorithm with the sparse example setting in mind. Motivations for the algorithm we present include the following:

[1] We use standard Prolog notation for constants and variables throughout this paper, but use standard logical notation otherwise.

1. High accuracy of the learned concept definition on teacher-labeled training examples.
2. Robustness in the presence of a small number of training examples and perhaps a total lack of negative examples.
3. Inclusion of most, if not all, teacher-mentioned predicates in the learned concept definition.
4. Flexible combination and generalization of the teacher's advice within and across examples.
5. Robustness to teacher errors, both in data labeling and advice.
6. Learned concepts may need to include predicates not mentioned in teacher-provided advice but supplied as part of example descriptions.

Our primary motivation is to allow human users of ILP systems to express their knowledge about the learning task at hand in whatever means seems most natural to them, from explaining (partially or fully) why some specific examples are positive or negative members of the concept being learned, to simply stating the proper categories (i.e., positive or negative) for other examples. We present our approach as a "batch" system that is given a set of labeled examples and possibly some advice about the examples, and then produces a set of one or more logical clauses ("inference rules") that best capture the concept being taught. However, we envision our approach as being best situated in a setting where the human-machine dialog is continual; the human teacher provides some initial training, the algorithm then learns, after which the teacher can provide additional guidance and the process repeats until an acceptable concept description results (where 'acceptable' can either be based on inspection of the learned clauses, or, more likely, on the quality of the predictions of the learned clauses for new examples).

ILP systems search a space of possible clauses composed from background knowledge; the space is typically defined declaratively by a set of mode definitions. For this work, we used an implementation that closely follows the Aleph ILP system [19], although we present general techniques with few Aleph dependencies. We made one major changes to the default Aleph implementation, wrapping Aleph in an iterative-deepening style search algorithm, as further explained in the next section.

As mentioned above, we address the problem of effectively incorporating, into the ILP framework, teacher-provided advice; a human teacher usually provides the latter and this interaction can be viewed from the wider perspective of human-machine interaction. Such teaching refers to humans teaching computers concepts and/or behaviors, through as *natural* and human-like dialog as possible. In our setting, the taught concepts take the form of logical theories and the teacher provides relevance advice about specific examples. The relevance advice takes a number of different forms, from simple "this feature is important" advice to complex statements that can be mapped to a grounded form of the concept being taught. The advice can be provided by a human familiar with the advice language but with no ILP experience, i.e., a non-expert. In the experiments we report later in this paper, all the instruction was provided by non-ILP-experts, all who are independent from this paper's authors.

Figure 1 illustrates, using propositional logic for simplicity, how advice-generated background knowledge can help focus ILP's search. A common ILP search strategy is to build clauses in a top-down manner, successively adding various literals that might

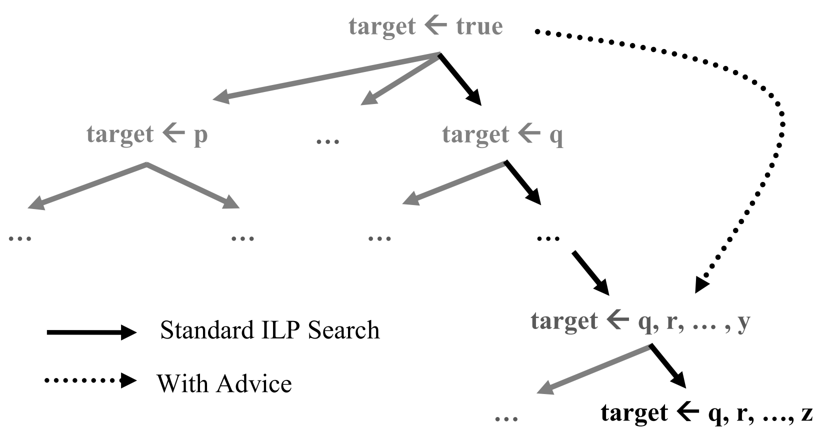


Fig. 1. An illustration of a top-down ILP search for a inference rule to predict the literal *predicate*, whose definition is the conjunction of literals *q* through *z*. Finding a long clause such as this can be quite hard, but if a teacher gives advice (possibly across multiple examples) that the conjunction of literals q through y is relevant, then finding the correct definition is much easier.

improve a rule. If a long clause is needed, the search space can be exponentially large, and if there are only a few training examples, many possible rules can accurately match the training examples. However, good background knowledge can quickly lead to the consideration of long clauses, as the figure shows. It can also help choose among many equally performing rules.

Section 3 discusses the conversion of this teacher-provided advice into ILP background knowledge. To accommodate differing amounts of advice, and different levels of concreteness of advice, we employ a control structure, explained in Section 2, which is capable of exploring successive layers of the hypothesis space, from a layer that tightly follows the advice to an outermost, rarely-used layer that effectively ignores the advice. Section 4 presents a discussion of our empirical study. Section 5 discusses the relationship to prior work within ILP.

2 The Onion

Our ILP advice algorithm generates a number of hypotheses, some of which we deem less likely than others. Additionally, since we assume the teacher has little to no ILP expertise, we must utilize a method to select ILP parameters automatically, in a way that consistently supports the motivations discussed above. We developed the *ONION*, a control structure that iteratively searches a set of successively larger and less likely ILP search spaces.

Algorithm 1 presents the basic structure of the ONION. Essentially, the ONION iterates over a set of ILP parameter settings and priority levels, performing an ILP search for each. When a given ILP search returns a theory, the ONION evaluates that theory against a set of acceptance criteria (such as minimum accuracy and coverage, i.e., precision and recall), and if the theory is acceptable, the ONION returns it.

Algorithm 1. THE ONION

```
1:  For each set of criteria C in {$C_1, C_2, \ldots, C_n$} for an acceptable theory
2:    For each priority level P in {High, Medium, Low, None}
3:      For each set of Aleph parameter settings S in {$S_1, S_2, \ldots, S_m$}
4:        Only consider literals or generated clauses whose relevance is a least P
5:        Call Aleph with parameter settings S
6:        If Aleph's learned theory meets criteria C then Return theory
7:  Return FAIL
```

We order the sets of acceptance criteria and possible settings from most restrictive to least restrictive, while we order the generated background according to a priority. We discuss how we assign that priority in the next section. The priority level "None" is given to those literals not mentioned by the teacher's advice, but which appear in the descriptions of examples.

3 Converting Advice to Background Knowledge

Teacher advice provides a method for the user to instruct our learning algorithm. The advice takes the form of logical statements. From this information, we construct new background knowledge representing sub-concepts. We also generate the necessary ILP modes (these modes specify the types of arguments and state, for the arguments of a new literal being considered for addition to a clause, which need already appear in the clause, which can be new variables, and which should be constants). Additionally, we attach priorities to all of the generated background knowledge for use by our ONION algorithm. Currently, we assume the teacher talks about a specific example (either positive or negative) and specifies the advice in a ground format that we then variablize into a general form. It is straightforward to extend our system to allow the teacher to provide generalized advice, but we believe that for most users it will be easier to explain why specific examples are or are not members of the concept being taught and that is the interaction style on which we focus.

Although we assume that the user understands basic logic (i.e., the meaning of AND, OR, and NOT), we attempt to allow the user to communicate advice in a natural, and possibly somewhat inaccurate manner. Thus, although we specify the exact logical format of the advice below, our system attempts to rectify common user misunderstandings, such as predicate/function confusion. We also do not expect the user to understand the algorithmic details of the underlying ILP system.

Table 1 shows two training examples and three pieces of teacher provided advice for a sample concept ReadyToFly. The ReadyToFly concept indicates, as one might guess, that an airplane is ready to fly. We will be using this simple concept to demonstrate our approach. We define the concept as:

$$\text{readyToFly(Plane)} \leftarrow \text{fueled(Plane)} \wedge \text{gearDown(Plane)} \wedge \neg\text{damaged(Plane)}.$$

Algorithm 2 details the process of creating background knowledge from the teacher-provided advice. The process proceeds in several phases. The first phase

Table 1. ReadyToFly concept. Training data includes two examples, one positive, one negative, along with three pieces of teacher provided advice, two pieces for the first example and one for the second.

Advice #	Ground Example	Pos/Neg	Teacher Advice
1	readyToFly(plane1)	Positive	fueled(plane1)
2	readyToFly(plane1)	Positive	gear_down(plane1)
3	readyToFly(plane2)	Negative	damaged(plane2)

(lines 4 to 10), variablizes the ground advice statements via applying anti-substitution, i.e., a mapping from occurrences of ground terms to variables. For our purposes, we only need to map constants to variables. The anti-substitution may be either a *direct-mapping* that maps all occurrences of the same constant to the same variable and occurrences of distinct constants to distinct variables, or an *indirect-mapping*, where occurrences of the same constant can be mapped to different variables.

Indirect-mappings address cases where two constants are coincidentally equivalent. This occurs regularly in examples with numeric constants, where common numbers such as 1.0 may perform two different roles. Later, when we assign priorities to generated background knowledge, those created with indirect-mappings receive a lower priority than those created with direct-mappings. Indirect mapping anti-substitutions perform what is sometimes called "variable splitting" in ILP [18], where two occurrences of the same term are generalized to different variables. It is well-known that variable splitting can lead to an increase in run-time that is exponential in the number of occurrences of the same term within a formula. In practice, such multiple occurrences are rare, except in the case of very common constants within a domain, for example, the 1.0 case discussed above. To prevent this exponential worst-case increase, in practice, we limit the maximum number of variable splittings (cases of two occurrences of the same term being mapped to distinct variables) by an anti-substitution to some small constant k. Alternative approaches to controlling the cost of variable splitting, such as employing domain-specific heuristics about commonly-occurring constants, are a direction for future research.

Table 2 depicts both a direct and indirect anti-substitution. As shown, we perform the same anti-substitution on both the example and the piece of advice, linking variables in the example to variables in the advice. Although not shown in Table 2 we generalize all advice for a single example at the same time. Thus, constants can be tied together across different advice for the same example, but are not tied across advice for different examples.

Table 2. Direct and indirect anti-substitutions. Direct anti-substitutions generalize equivalent term to the same variable. Indirect anti-substitutions generalize equivalent terms to different variables.

Ground Example & Advice	Anti-substitution	Type
readyToFly(plane1) $\leftarrow$ fueled(plane1)	readyToFly(X) $\leftarrow$ fueled(X)	Direct
	readyToFly(X) $\leftarrow$ fueled(Y)	Indirect

Algorithm 2. GENERATEBACKGROUNDKNOWLEDGE

```
1:  Given: Labeled examples, some of which have associated advice
2:  Do:    Infer generalized background knowledge
3:
4:  For each example $e_i \in \{e_1 \ldots e_n\}$
5:    Given advice $A_i$ associated with example $e_i$
6:    if $e_i$ is positive example then create an associated implication $e_i \longleftarrow A_i$
7:    else create an associated implication $e_i \longleftarrow \neg A_i$
8:
9:    Generate all non-equivalent formulas via anti-substitution
10:      from the implication to yield the set of formulas $F_i$
11:
12: Let F denote the set $F_1 \cup F_2 \cup \ldots F_n$
13: Standardize apart all formulas in F
14: Let $\theta$ be the most general unifier of all consequents of formulas in F
15: For each $F_i$
16:   Apply $\theta$ to all formulas in $F_i$ to yield $F'_i$
17:   Collect all antecedents from formulas in $F'_i$ to yield $G_i$
18:
19: Let $H = \{\}$, a set of generated rule antecedents
20: For each generalized advice-piece $G_i$
21:   Let $H = H \cup G_i$   // Per-piece antecedents
22:
23: For each example $e_j \in \{e_1 \ldots e_n\}$ with associated advice
24:   Let $K_j = \cup \{ g \in G \mid g$ was generated from example $e_j$ advice$\}$
25:   Let $H = H \cup K_j$   // Per-example antecedents
26:
27: Let $H = H \cup \{$ "Mega-Rules" $\}$   // See text
28:
29: For each generated logical combination $h \in H$, introduce a new predicate
30:  p and assert $p(V_1, V_2, \ldots, V_k) \longleftarrow h$, where $V_1, V_2, \ldots, V_k$ are variables
31:  generalized from constants in the example literal
32:   If $p \longleftarrow h$ is a Mega-Rule then assign $p \longleftarrow h$ High priority
33:   else if $p \longleftarrow h$ is per-example then assign $p \longleftarrow h$ Medium priority
34:   otherwise assign $p \longleftarrow h$ Low priority
35:
36: Return set of all generated implications $p \longleftarrow h$ along with priorities
```

Given the generalizations from the first phase, the second phase of GENERATEBACKGROUNDKNOWLEDGE (lines 12 to 17) performs a unification to merge variables that arose from constants found in the examples themselves. For instance, if we consider the direct anti-substitutions of all pieces of advice, we have *readyToFly(A) ⟵ fueled(A); readyToFly(A) ⟵ gear_down(A);* and *readyToFly(B) ⟵ damaged(B).* After the unifications, the implications would be *readyToFly(X) ⟵ fueled(X); readyToFly(X) ⟵ gear_down(X); and readyToFly(X) ⟵ damaged(X)* where X is shared. This allows distinct constants from different examples that played the same role to be merged into a single variable.

In the third phase (lines 19 to 27), we generate compound logical formulas by connecting the generalizations for different pieces of advice with the AND and OR logical connectives. We generate three different styles of formulas: per-piece, per-example, and "Mega Rules". The per-piece formulas correspond to the individual pieces of advice specified by the teacher. The per-example formulas aggregate all the advice provided for a single example into one formula with the individual pieces of advice joined via AND connectives. The per-example formulas allow the teacher to provide many small pieces of advice about an example instead of requiring the teacher to compose a single complex piece of advice. Finally, Mega Rules attempt to capture all of the advice into one logical statement, by conjoining direct-mapping generalizations of all advice from all positive and negative examples.

More specifically, we do the following to produce our Mega Rules:

> Let F_i be the logical formula that our algorithm produces by conjoining ("ANDing") all of the relevance statements about positive example i.[2]
>
> Let G_j be the logical formula that our algorithm produces by conjoining all of the relevance statements about negative example j.
>
> We make the following Mega Rules, where i ranges over the positive examples with advice and j over those negative examples with associated advice:

$$(F_1 \wedge \ldots \wedge F_i) \wedge \neg(G_1 \vee \ldots \vee G_j) \rightarrow example$$
$$(F_1 \wedge \ldots \wedge F_i) \wedge \neg(G_1 \wedge \ldots \wedge G_j) \rightarrow example$$
$$(F_1 \vee \ldots \vee F_i) \wedge \neg(G_1 \vee \ldots \vee G_j) \rightarrow example$$
$$(F_1 \vee \ldots \vee F_i) \wedge \neg(G_1 \wedge \ldots \wedge G_j) \rightarrow example$$

The above are all ways to explain a collection of teacher-provided advice, though some are more natural than the others. In the first one, our algorithm interprets the teacher as using each positive example to provide aspects of a conjunctive concept and each negative example to state properties that members of the concept lack ("this is a bird because it has wings, this other example is a bird because it lays eggs, this third example is not a bird because it has leaves, this fourth example is not a bird because it is made of metal. …"). The third and fourth lines are appropriate for disjunctive concepts ("Alice got to work by taking the bus. Bob got to work by walking. … Carl did not make it to work because he slept all day."). In addition to the rules shown above, we also generate four additional Mega Rules in which we negate positive advice and do not negate the negative advice.

When only direct-mapping generalizations exist, only a handful of formulas are generated, providing excellent scalability. When indirect-mappings occur, we generate additional rules in which we substitute all combinations of the indirectly-mapped advice pieces into the per-piece and per-example formulas. This process scales exponentially in the number of indirect-mapping generated.

[2] We conjoin statements about the same example because we assume the teacher is telling us various properties of that example that all hold, though it would be reasonable to extend our algorithm by disjunctively combining them as another alternate interpretation. By allowing only one or two ways to combine advice about the same example, we avoid the combinatorics that would arise by combining in all logically possible ways.

Table 3. Generated Background Knowledge. Three types of background knowledge are created during advice processing: per-piece, per-example, and mega-rule background knowledge. Per-piece is composed of single pieces of advice. Per-example is composed of all advice piece for a single example. Mega-rules use all provided advice combined via various logical operators.

Generated Background Knowledge	**Type**	**Priority**
readyToFly(X) $\leftarrow$ fueled(X) $\wedge$ gear_down(X) $\wedge \neg$damaged(X)	Mega-Rule	High
readyToFly(X) $\leftarrow$ (fueled(X) $\vee$ gear_down(X)) $\wedge \neg$damaged(X)	Mega-Rule	High
readyToFly(X) $\leftarrow$ fueled(X) $\wedge$ gear_down(X)	Per-Example	Medium
readyToFly(X) $\leftarrow \neg$damaged(X)	Per-Example	Medium
readyToFly(X) $\leftarrow$ fueled(X)	Per-Piece	Low
readyToFly(X) $\leftarrow$ gear(X, down)	Per-Piece	Low

In the final phase (lines 29 to 34) we convert each of the generated formulas back into an implication (as a precursor to creating ILP background knowledge). During this phase, we assign a search priority with a preference for longer formulas, i.e., those that use as much of the user-provided advice as possible. Mega Rules receive the highest priority, followed by per-example formulas, and finally per-piece formulas. The ONION uses these priorities to order the ILP search.

Table 3 shows several of the implications generated for the sample concept. The head of the generated clause exposes all of the variables from the logical formula. Variables tied to the example during the generalization phase become input variables for the clause. In many situations, it is also advantageous to expose variables occurring in the body of the rule as output variables. However, exposing additional variables increases the size of the ILP search space. In absence of other information, for each formula, we expose only a single output variable. For any given formula, we determine the output variable by considering all of the literals that we derived from positive pieces of advice and selecting the last variable that occurs. This approach scales well. However, in some cases, variables that would be helpful may not be exposed in the head of the generate clause. Clauses derived from formulas with OR connectives have the additional requirement that the selected output variable must occur in all of the OR-ed subformulas. Determining whether a variable occurs in all of the subformulas requires us to determine if two variables are equivalent. If argument-type information is available, we require only that type of the output variable match in all of the subformulas. In the absence of typing information, we disallow output variables for disjunctive formulas.

Statements about negative examples can be ambiguous. Imagine a teacher says an example is negative because *color = blue*. Does this mean the example is negative because it is blue or because it is not? Because the teacher is talking about specific examples that are observable by our learning algorithm, we address this in an obvious way. Namely, we evaluate the teacher's statement on the current example, and we then, if necessary negate the advice so that it says something that is true. Hence if the current example is red, we standardize the advice about color to *color* $\neq$ *blue*.

Finally, we assign input variables both an ILP input mode of '+' (the argument must already be in the clause being constructed) and a constant mode of '#', plus we assign output variables both an output mode of '–' (a new variable can be introduced) and a constant mode of '#'. Additionally, our algorithm also works when the ground advice contains logical functions. We convert the functions into Skolem-constants and perform generalizations over all possible combinations of the Skolem-constants.

4 Experiments

We performed several experiments to demonstrate the performance of the ONION with and without advice. In addition to measuring learning on our test beds, we also conducted comprehensive empirical analyses to study the performance of the ONION when there is noise in the labels on examples, as well as in the advice. These experiments are designed to demonstrate the effectiveness of the ONION in the absence of advice, its robustness to noise, and how the system is capable of generalizing advice about specific examples to all the available examples leading to improved learning and accuracy. The improvements in generalization performance can be significant, especially in the presence of a very small number of examples.

4.1 Test Beds

We used learning tasks developed by an independent third party under the Bootstrap Learning (BL) project [12] funded by the United States Defense Advanced Research Projects Agency (DARPA). In the BL setting, the machine learner induces concepts that build upon one another through a "ladder" of tasks, which are organized as self-contained *lessons*; lower rungs of the lesson ladder teach simpler concepts, which are learned first and then used to learn – i.e., *bootstrap* – more complex concepts. The lessons in the project incorporate a wide variety of natural teacher instruction methods, including providing domain descriptions, pedagogical examples, telling of general instructions, demonstration, and feedback. Our role in the project is supervised learning from examples. For our experimental setup considered here, we use 14 lessons from two domains of the BL project: Unmanned Aerial Vehicle (UAV) and Armored Task Force (ATF).

For each of the 14 lessons, third-party domain-experts, under the direction of DARPA and not under our control, generated "lessons" to teach these tasks. The lessons consist of a sequence of *messages* from the teacher to the learner. Teacher instruction includes providing *training examples* (up to 100 examples for each task) and expert advice for certain examples to help the student learn these tasks effectively. For each lesson, we wrote software that converted the messages into ILP facts and examples expressed in predicate calculus. The mean accuracy of always guessing the majority category across each of these 14 lessons is 57%.While we had access to the UAV and ATF testbeds during algorithm development, during Fall 2009 our algorithm was applied by DARPA to a "hidden" testbed, to which we had no access. Our approached produced 100% accuracy on learning in that testbed. Since a variant of that testbed will be used Spring 2011, at the time of this writing we have no knowledge of the testbed and, hence, cannot report anything more about it here.

Methodology. We are interested in studying the behavior of our advice algorithm, along with the Onion, with respect to several criteria: (1) its ability to learn diverse concepts across domains without the intervention of an ILP expert, (2) its ability to effectively exploit teacher advice in order to learn concepts with only a small number of examples, (3) its robustness to teacher errors of commission in the examples (mislabeled examples) and (4) its robustness to teacher errors of omission in the advice (incomplete or missing literals). Our experimental study consists of three experiments that we describe below. For each lesson we have 100 training examples and 100 test set examples. During our experiments, we split the training set to generate a tuning set, used by the Onion for evaluating parameter settings. For runs with more than 25 examples, we place two-thirds of the data provided to our learner into a training set and one-third the data into a tuning set. For runs where fewer than 25 training examples, we do not use a separate tuning set, instead relying directly on training-set accuracy to tune parameters in the Onion. In all experiments there were an equal number of positive and negative examples.

Because there is an intended pedagogical order to the examples, some of which have associated advice, we did not perform 10-fold cross validation within each lesson (in addition, since we have data simulators, cross validation is not necessary – instead we simply use fresh samples of 100 examples as our test sets). The results presented for each experiment are the test-set accuracies averaged over all 14 tasks.

Across all of these tasks, we used the same parameter choices in the Onion. That is, over all of the experiments that we report here, our ILP system was run *unchanged.* Our Onion approach was able to find good parameter settings, trading off computer time for the ability to operate without intervention from an ILP expert.

4.2 Results and Discussion

Experiment A. In our first experiment, we compare the performance of the Onion with and without advice over all the 14 tasks. Figure 3 shows the results, where we plot *learning curves*, i.e., test-set accuracy as a function of increasing numbers of training examples. (As mentioned earlier, our implementation is not an on-line, incremental learner. We simply run in “batch mode” for various numbers of training examples.).

In the case where the learner is not given any advice, the Onion is able to generalize across tasks and domains, and obtain an average test-set accuracy of 74.0% when using all 100 training examples. Even when using smaller fractions of training data, the Onion is able to effectively select parameters and automate the setup task to obtain learning rates in excess of 57%, which is equivalent to random guessing. The main results in Figure 3 however, are the test-set accuracies achieved by the Onion in the presence of advice. Even when using only four training examples per lesson, the Onion with advice achieves an average test-set accuracy of 93.8%, and reaches 100% with only ten examples.

Experiment B. Experiment A involved advice from a 3rd party who was careful to create rich and accurate advice. However, real teachers are likely to make errors. In our 2nd experiment we simulate *errors of omission* by dropping literals from advice.

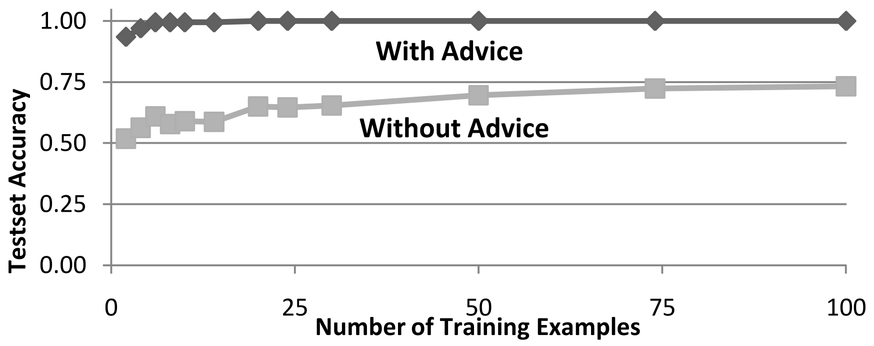


Fig. 3. Experiment A: Testset accuracy as a function of the number of training examples ("learning curves"), with and without advice

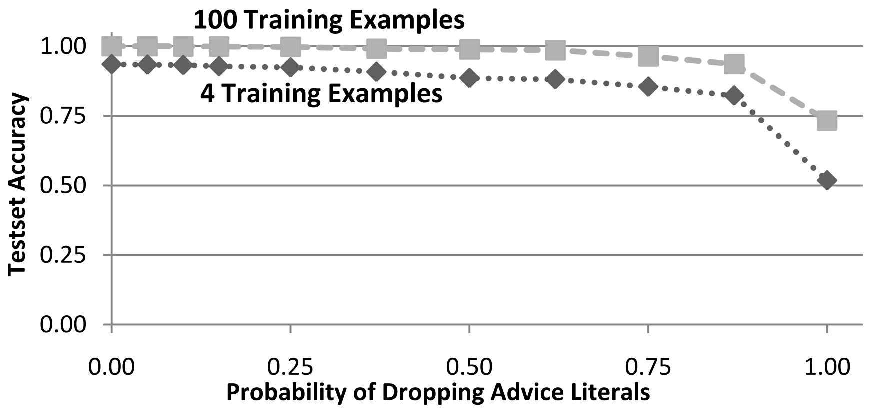


Fig. 2. Experiment B: Impact of errors of omission in advice. The x axis indicates the probability value used in the advice-removal process (see text).

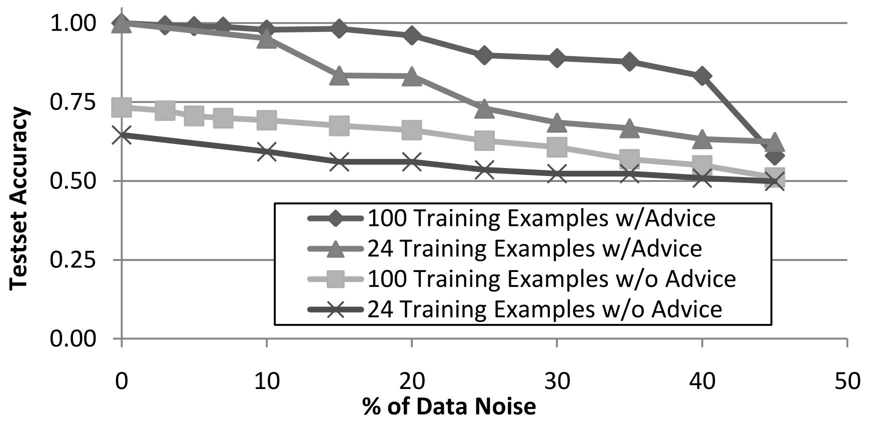


Fig. 4. Experiment C: The impact of mislabeled examples under various conditions. The *x*-axis indicates the percentage of training data that is mislabeled.

We randomly drop literals as follows. For each advice rule we flip a weighted coin, and it if comes up 'heads' we delete the *last* literal in the rule. If we deleted the last literal, we flip the coin again and consider deleting the second-from-last literal; this continues until either the rule's literals are exhausted or the coin comes up 'tails.' In the later case, we place the possibly truncated advice rule in our "noisy" advice set. We choose to remove from the end of advice rules, since prefixes of conjunctive rules are likely to be partially coherent, whereas dropping literals from the middle of multi-literal (i.e., conjunctive) statements may lead to nonsensical advice. A topic for future research is to create more realistic models of imperfect advice.

Figure 2 shows the results of our errors-of-omission experiment, where we plot the test-set accuracy as a function of the probability of randomly removing each literal as specified above. For each selected probability-of-removing, we generated 30 independent "noisy" advice sets for each of our 14 lessons. The impact of noisy advice depends on the number of training examples, so we perform this experiment using 2 and 100 training examples.

The behavior of the system with different training set sizes is nearly identical with the test-set accuracy dropping steadily as increasing fractions of advice literals are removed. However, with advice-omission rates as high as 50%, and with a small number of examples, the ONION is able to produce average test-set accuracies of over 80%. This demonstrates that even partial advice can be effective for learning, and that the ONION is able to leverage this information effectively even in the presence of significant imperfections in advice.

Experiment C. In this final experiment, we compare the performance of the ONION with and without advice in the presence of mislabeled training examples. Figure 4 shows the results; we plot test-set accuracy as a function of the percentage of mislabeled examples. We generate the noisy examples by first removing examples that have advice attached from the set of training examples. With the remaining training examples, we randomly select a fraction of the examples and flip their labels. We then replace the examples with attached advice into the training set. Care was taken to guarantee that the final fraction of mislabeled examples was correct when the examples with advice added back into the training set. This approach to noise generation limited the range of noise available, especially for small training sets. For instance, if we have a training set size of four and two of those were examples with advice attached, the minimum amount of noise that can be considered is 25% (the result of flipping a single, non-advice, example).

For experiment C, we generate 30 independent sets of mislabeled examples and, separately, ran with and without advice using each of these noisy data sets. The results are averaged over all random 30 runs and over all 14 tasks. As expected, the example noise reduces the performance of both the advice-free and with-advice cases. The main result from Figure 4 is that our advice algorithm, combined with the ONION, performs well even in the presence of large amounts of data noise. In contrast, the no-advice case degrades more quickly to about the level of random guessing (57%).

In summary, our experiments demonstrate that our advice-taking ILP system can learn well from advice about a small number of specific examples, while being robust to errors in advice and example labels. They also show that our ONION approach can be an effective method even when users provide no advice.

5 Related Work

ILP research has a rich history of developing systems capable of initiating human-computer interaction and using them guide and constrain the search. The most notable such systems include MARVIN [15], MIS [16], DUCE [10], CIGOL [9] and CLINT [2] where the algorithm can ask the human one or more questions that would guide the search. For example MIS relied on the human answering queries by providing the labels of examples, together with a proof, or derivation, for each positive response. In contrast, in our present work the human initiates the input by providing advice, either in general or in association with the original training data.

Another general area of related work is theory refinement or theory revision (e.g., [7, 13, 19, 20]) where the user provides an initial logical theory that explains many and not all examples, and the learning system must modify this theory. As a result the search is constrained to prefer theories close to the original theory, similar to the present work. But a key distinction is that the advice in our present work is *example-specific*, which can substantially ease the burden on the user, as compared to expressing abstract rule(s) underlying the concept.

Our work is closely related to argument-based machine learning (ABML [8]) that takes as input user-provided advice about specific examples, in the form of an *argument*. A key distinction is that the present work does not assume the arguments are exactly correct and therefore may combine various pieces of different arguments in order to construct rules. Another distinction is that to our knowledge ABML has been applied strictly to propositional-rule learning.

Automatic parameter selection for machine learning methods has been explored earlier [5], where the goal is to use the expected error for each parameter setting to guide the selection of the parameters for decision trees. Lavrac et al. [6] proposed a feature selection framework for ILP that worked well in propositional learning and special cases of relational learning. This was later extended by Alphonse and Matwin [1] using powerful statistical feature-selection techniques to control the dimensionality of the search space. The key idea in their work is to reduce ILP examples to non-recursive Datalog clauses by removing irrelevant literals. Muggleton [11] theoretically shows that as the number of predicates in the background theory increases, the size of the search space of an ILP system can increase greatly. This necessitates the intervention of an ILP expert who can reduce the search space. In such a context, relevance information becomes crucial. Srinivasan et al. [17] conducted an empirical study in several biomedical domains and concluded that when not all the background knowledge can be used, the relevance information from the expert is very useful in construction of good domain models.

6 Conclusions and Future Work

Not surprisingly, teacher advice is useful to learning. The key challenge is the need to generalize hints and advice the teacher gives about specific examples so that it accurately applies to future examples. We present a formal approach to incorporating this into the wider framework of ILP. The empirical results show that our system is able to learn well, across multiple concepts, from a combination of training examples and teacher-provided hints. Running our ILP system without these hints – i.e., only using the training examples – also produces reasonable accuracies on held-out ("test set") examples. Another key challenge is effective parameter selection and the automation of the ILP-setup task. The final challenge is to ensure that the system is robust to noise, both in examples and in advice.

We evaluated our algorithms, holding all default parameter settings constant, on 14 tasks from two domains designed by third-parties not under our control; these human teachers provided training examples as well as relevance information. In our experiments, we demonstrated that our system, the advice-taking Onion, is capable of (1) effectively automating the ILP-setup task over different tasks from significantly different domains, (2) exploiting teacher hints and relevance information to learn concepts with near-perfect test-set accuracies even if given only a small set of training examples, (3) performing effective parameter selection to learn concepts well when there is no teacher advice, (4) being robust to example-label noise that can arise from teachers' errors of commission, and (5) being robust to advice noise that is likely to arise from teachers' errors of omission.

Currently, we are focusing on improving our layered approach, to more robustly automate ILP in tasks that are more complicated. Also, we are currently looking at further exploiting teacher-provided feedback beyond statements about which features and objects are relevant, such as allowing teachers to provide corrections to previous advice statements. A possible future direction is to explore the possibility of refining the learned theories using teacher feedback in the lines of theory refinement for ILP [8, 11, 12, 13, 14]. Refining teacher's advice is important as it renders the ILP systems more robust to teacher errors that occur naturally in human teaching. Another future direction is deploying our approach in the context of probabilistic-logic learning [4]. A final appealing direction of this research is to embed it into some user interface where a human can train their software by a combination of making simple English statements, pulling down menus and selecting items, and gesturing at objects (e.g., clicking with the mouse, a pen, or even one's finger) to indicate relevance and objects of discourse ("this object should not be near that one").

In this work, we considered the problem of simplifying the use of ILP for non-ILP experts. More precisely, we considered a human teacher who is trying to teach an ILP learner using a mixture of examples and advice about these examples. We outlined the challenges and presented solutions for the generation of background knowledge from teacher advice, utilizing a layered ILP-search approach.

Acknowledgements

The authors gratefully acknowledge the support of the DARPA's Bootstrap Learning program via the United States Air Force Research Laboratory (AFRL) under grant HR0011-07-C-0060. Views and conclusions contained in this document are those of the authors and do not necessarily represent the official opinion or policies, either expressed or implied, of the US government, DARPA, or AFRL.

References

1. Alphonse, E., Matwin, S.: Feature subset selection and inductive logic programming. In: Proceedings of the 19th Intl. Conf. on Machine Learning, pp. 11–18 (2002)
2. De Raedt, L.: Interactive Theory Revision: An Inductive Logic Programming Approach. Academic Press, London (1992)
3. Finn, P., Muggleton, S., Page, D., Srinivasan, A.: Discovery of pharmacophores using the inductive logic programming system Progol. Machine Learning 30, 241–270 (1998)
4. Getoor, L., Taskar, B. (eds.): Introduction to Statistical Relational Learning. MIT Press, Cambridge (2007)
5. Kohavi, R., John, G.: Automatic parameter selection by minimizing estimated error. In: Proceedings of the 12th International Conf. on Machine Learning, pp. 304–312 (1995)
6. Lavrac, N., Gamberger, D., Jovanosk, V.: A study of relevance for learning in deductive databases. Journal of Logic Programming 40, 215–249 (1999)
7. Mangasarian, O., Shavlik, J., Wild, E.: Knowledge-based kernel approximation. Journal of Machine Learning Research 5, 1127–1141 (2004)
8. Mozina, M., Zabkar, J., Bratko, I.: Argument based machine learning. Artificial Intelligence 171, 922–937 (2007)
9. Muggleton, S., Buntine, W.: Machine invention of first-order predicates by inverting resolution. In: Proceedings of the 5th Intl. Conf. on Machine Learning, pp. 339–352 (1988)
10. Muggleton, S.: DUCE, an oracle based approach to constructive induction. In: Proceedings of the International Joint Conf. on Artificial Intelligence, pp. 287–292 (1987)
11. Muggleton, S.: Inverse entailment and Progol. New Generation Comp. 13, 245–286 (1995)
12. Oblinger, D.: Bootstrap learning - external materials (2006), `http://www.sainc.com/bl-extmat`
13. Pazzani, M., Kibler, D.: The utility of knowledge in inductive learning. Machine Learning 9, 57–94 (1992)
14. Richards, B., Mooney, R.: Automated refinement of first-order Horn-clause domain theories. Machine Learning 19, 95–131 (1995)
15. Sammut, C.: Learning Concepts by Performing Experiments. Ph.D. Dissertation, Department of Computer Science, University of New South Wales (1981)
16. Shapiro, E.Y.: Algorithmic Program Debugging. MIT Press, Cambridge (1983)
17. Srinivasan, A., King, R.D., Bain, M.E.: An empirical study of the use of relevance information in inductive logic programming. JMLR 4, 369–383 (2003)
18. Srinivasan, A., Muggleton, S., King, R.: Comparing the use of background knowledge by inductive logic programming systems. In: Proc. 5th ILP Workshop (1995)
19. Srinivasan, A.: The Aleph Manual, `http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/aleph.html`
20. Towell, G., Shavlik, J.: Knowledge-based artificial neural networks. Artificial Intelligence 70, 119–165 (1994)
21. Walker, T.: Broadening the Applicability of Relational Learning. Ph.D. Dissertation, Computer Sciences Department, University of Wisconsin – Madison (forthcoming, 2011)

Speeding Up Planning through Minimal Generalizations of Partially Ordered Plans

Radomír Černoch and Filip Železný

Czech Technical University in Prague
{cernorad,zelezny}@fel.cvut.cz

Abstract. We present a novel strategy enabling to exploit existing plans in solving new similar planning tasks by finding a common generalized core of the existing plans. For this purpose we develop an operator yielding a minimal joint generalization of two partially ordered plans. In three planning domains we show a substantial speed-up of planning achieved when the planner starts its search space exploration from the learned common generalized core, rather than from scratch.

1 Introduction

Automated planning has been a core area in artificial intelligence research for decades [1]. Nevertheless, novel planning algorithms are still being designed and several international annual competitions demonstrate the perpetual improvement in their performance. Of course, this unceasing progress also indicates the persisting room for improvement. To this end, a lively research direction aims at improving a planner by exploiting experience obtained in previously completed planning tasks. This general strategy has a clear motivation in that a typical deployment of a planner is in repetitive planning tasks only slightly varying in their initial conditions and desired goals. The specific approaches to exploiting previous plans differ mainly in two respects: what kind of knowledge they extract from the old plans, and what techniques they employ for that extraction.

As for the former aspect, a popular state-of-the-art approach is to look for structures (e.g. sequences) of actions that are frequently found in the plans [2–4]. Such actions are then glued into a single *macro-operator* made available to the planner. In other approaches [5, 6], *control rules* are extracted serving as heuristics for the choice of a suitable action given the preceeding actions already in the plan.

As for the latter aspect, techniques for analyzing previous plans may roughly be projected onto a spectrum between the 'deductive' and 'inductive' extremes. An example of the deductive approaches is [4] which determines the dependence structure in plans and suggests action sets which can act as macro-operators. Halfway in the spectrum are algorithms which search frequent action-set patterns in a narrowly prescribed form [3]. Lastly, established machine learning algorithms have also been explored in this context, such as for learning Markov chains for probabilistic description of action sequences [6].

Here we are mainly interested in learning from plans using the expressive formalism of first-order logic. This is a natural choice since planning problem descriptions and plans themselves are typically encoded in fragments of first-order logic. Previous research in this direction explored the application of explanation based learning [7]. Also ILP applications in the planning area have been reported, dating back to early works on relational reinforcement learning [8]. ILP has also been employed for the already mentioned standard task of learning control rules [9, 10].

By commonsense assessment of all the above reviewed approaches, we think they share one important deficiency. If we apply a planning algorithm repeatedly with only slightly varying task descriptions, we may expect the resulting plans to also vary only slightly. In other words, they may share a very large core structure, perhaps even containing the majority of actions inside the plans. Under such circumstances, it is clearly underambitious to use old plans only for seeking small nuggets such as control rules or search heuristics. Indeed, it is more adequate to directly determine the entire common core of the plans rather than just learn heuristics to guide their reconstruction. Finding such largest common cores is the goal pursued by the current study. The way such findings will be exploited by the planner is also different from the current approaches. Rather than producing new macro-operators or control rules, the discovered core plan structure will directly be used as an initial incomplete plan to be refined by the planner towards satisfying all initial conditions and goals of the current task. In other words, rather than equipping the planner with new operators or heuristics, we will advise a specific place in its search space where to begin the plan refinement.

Finding shared plan cores does not simply correspond to detecting the largest set of actions found in all input plans. Roughly, there are two factors making the problem more complicated and interesting. First, we must respect action dependencies reflected by a partial order defined on each input plan. Second, in learning a plan core, we should be able to abstract e.g. from domain-specific object names by means of variables. The presence of a candidate structure in a plan will thus be checked in a way more resembling θ-subsumption than the subset relation.

In terms of description complexity, plan cores will obviously be larger than learned control rules or heuristic functions. It is unrealistic to expect that *top-down* (general-to-specific) learning approaches would scale to searching among candidate structures possibly containing tens or hundreds or more actions. We therefore base our approach on a *bottom-up* strategy where input plans are jointly generalized. The central operator we develop and employ for this purpose is one that produces a minimal joint generalization of two partially ordered plans. The operator is obviously inspired by Plotkin's least general generalization [11] of clauses. Unlike operators previously developed for totally ordered clauses [12, 13], here we work with partial orders.

The paper is organized as follows. In the next section we explain the method for the joint generalization of partially ordered plans. In Section 3 we evaluate the method empirically and in Section 4 we conclude the paper.

2 Method

A plan is simply a partially ordered set of actions, where an action is a first-order atom such as $\mathsf{pickup}(\mathsf{block}, \mathsf{lefthand})$.

Definition 1. *A plan $P = (A, \leq)$ consists of a set of atoms $A = \{a_1, ..., a_n\}$ and a reflexive, transitive and antisymmetric relation $\leq$ on A.*

Note that terms in actions need not be ground. In fact, in what follows we specifically aim at generalizing plans and in so doing, we will turn constants into variables. Such generalized non-ground plans will act as a starting point for further refinements conducted by a planner in a new learning task, and variables will be grounded only as a result of these refinements.

Further in the text we will denote the transitive reduction of $\leq$ as $\leq^0$. Using the antisymmetry of $\leq$, the transitive reduction $\leq^0$ is always unique. We now proceed to defining a generality order on plans using the well known concept of OI-substitution [14], in which the variable-term mapping is injective.

Definition 2. *Let $K = (A_K, \leq_K)$, $L = (A_L, \leq_L)$ be plans. Plan K subsumes plan L iff there is an object-identity (OI) substitution θ such that*

$$A_K\theta \subseteq A_L \text{ and } \leq_K^0 \theta \subseteq \leq_L^0 \ .$$

We will denote the subsumption relation as $K \rhd_\theta L$.

Example 1. Consider the following plans:

$$L = (\{m = \mathsf{pick}(\mathsf{box}), n = \mathsf{move}(\mathsf{a,b}), o = \mathsf{drop}(\mathsf{box})\}, \{m \leq n, n \leq o\})$$
$$K_1 = (\{p = \mathsf{pick}(Z), q = \mathsf{move}(\mathsf{a,b})\}, \ \{p \leq q\})$$
$$K_2 = (\{r = \mathsf{pick}(X), s = \mathsf{drop}(Y)\}, \ \{r \leq s\})$$

Obviously we can see that $K_1 \rhd_\theta L$, where $\theta = \{Z \backslash \mathsf{box}\}$. Then note that $K_2 \not\rhd_\theta L$, because subsumption is defined through the transitive reduction $\leq^0$ rather than the partial order $\leq$ itself.

The reason for relying on OI-substitution as opposed to its standard counterpart is straightforward. If we had used standard substitution, several actions in a generalized plan could correspond to a single action in the more special (training) plans. This would be counter-intuitive: if all training plans contain a pick action only once, we do not want the generalized plans to contain this action multiple times (e.g., contain both $\mathsf{pick}(\mathsf{box}, X, \mathsf{robot})$ and $\mathsf{pick}(\mathsf{box}, \mathsf{room}, Y)$).

Example 1 emphasizes the fact that the definition of $\rhd$ subsumption uses the transitive reduction $\leq^0$ rather than the transitive relation $\leq$ itself. The reason is that generalized plans should not contain two preceding actions regardless of what happens in-between. Instead we seek sets of actions, which are executed in a block without being interleaved by any other actions.

Using the generality order we now define common generalization of plans.

Definition 3. *Let G, K, L be plans. Plan G is a* generalization *of K and L ($G = K \diamond L$) iff $G \rhd_\theta K$, $G \rhd_{\theta'} L$ and the order of generalized actions is preserved:*

$$\forall a, b \in A_G.\ (a \leq_G b) \rightarrow (a\,\theta \leq_K b\,\theta) \wedge (a\,\theta' \leq_L b\,\theta')$$

The generalization G of K and L is called a minimal generalization iff $G = K \diamond L$ and there is no G' such that $G \neq G'$, $G \rhd G'$ and $G' = K \diamond L$.

Example 2. In general, there can be multiple minimal generalizations of two plans. Consider the plans

$$K = (\{\mathsf{P(a,b)}, \mathsf{P(c,d)}\},\ \ \mathsf{P(a,b)} \leq \mathsf{P(c,d)})$$
$$L = (\{\mathsf{P(a,d)}\}, \emptyset)$$

for which we want to find minimal generalizations. If $\mathsf{P(a,d)}$ from L is identified with $\mathsf{P(c,d)}$ from K, their minimal generalization is $G_1 = \mathsf{P}(X, \mathsf{d})$. Alternatively we can identify $\mathsf{P(a,d)}$ with $\mathsf{P(a,b)}$, which gives a minimal generalization $G_2 = \mathsf{P}(\mathsf{a}, Y)$. Note that neither $G_1 \rhd G_2$ nor $G_2 \rhd G_1$ and no more specific generalizations can be found. Hence both G_1 and G_2 are minimal generalizations of K and L.

This process described above will give rise to the the definition of least generalization with respect to a given mapping[1] between atoms of generalized plans.

Definition 4. *Let K, L, G be plans. Let there be an injective partial function $f : A_K \rightarrow A_L$. Then we say that G is the least generalization of K and L with respect to f iff $G = K \diamond L$, $G \rhd_\theta K$ and $G \rhd_{\theta'} L$, where θ and θ' are obtained from the anti-unification algorithm on pairs $(k, f(k))$, where $k \in \mathrm{domain}(f)$.*

The definition of least generalization with respect to a mapping is practical for finding minimal generalizations. It reduces the task to finding a correspondence between atoms in plans, because the θ substitutions are obtained from the deterministic anti-unification algorithm. Relying on the above introduced concepts, we designed an algorithm for finding minimal generalizations of 2 plans (K and L):

1. The essential phase constructs the mapping $f : A_K \rightarrow A_L$ with a *depth-first search* approach. Initially f is empty. A pair of actions from K and L is added to f if they share the predicate symbol (which represents the action name). Then the relations $\leq'_{\{K,L\}}$ are constructed as subsets of $\leq_{\{K,L\}}$ on all atoms present in $A_{\{K,L\}}$.
   If these relations satisfy the conditions in Definitions 2 and 3, the algorithm iterates with the newly constructed f. Otherwise the new f is discarded and the algorithm backtracks.
   The first phase ends if it is not possible to add any more actions into f.
2. The atoms A_G in the generalized plan G are obtained from the *anti-unification* algorithm on pairs of atoms in f.
3. The relation $\leq_G$ is taken directly from $\leq'_K$ and $\leq'_L$. Note the given the conditions in Definitions 2 and 3, both relations should have the same structure.

[1] The mapping corresponds to the concept of *selection* used in ILP literature.

3 Experiments

Here we test whether our plan-generalization approach allows to reduce planning times when compared to two baseline planning regimes.

Materials. In all experiments, we employ the *Plan-space algorithm* [1] as a representative of planning algorithms using the classical *STRIPS* representation based on first-order logic. Plan-space is a backward-chaining planning algorithm with the "least-commitment strategy": 2 actions are not ordered unless they are required to. Hence the produced plan is a partially-ordered set of actions unlike the case of state-space planning.

We have used 3 different planning domains (gold-miner, rover, and gripper) each containing a single "etalon" problem. A *problem* is a tuple (I, G) where I is the set of initial conditions and G is the set of goals. The etalon problem for domain d is denoted (I_d, G_d). As further experimental material, we use random planning problems (I, G) generated from the etalon problems in such a way that $I = I_d$ while G contains goals randomly sampled from G_d. Given that the etalon problems are solvable, it follows that these randomly generated problems are also solvable.

Table 1. Properties of the *etalon problem* in all domains

Domain	Initial propositions	Goal propositions	Plan length	Time to solve
`gripper`	9	7	9	7m
`goldminer`	26	5	9	42m
`rover`	18	8	8	1h 15m

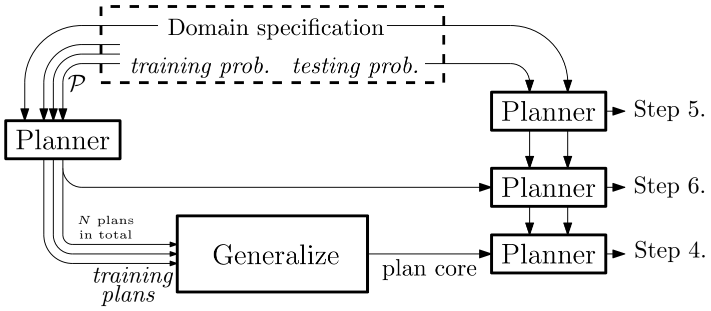


Fig. 1. Workflow of the testing process

Protocol. We tested our approach using the following workflow (refer to Fig. 1) reflecting its anticipated use. For each domain d, each $S \in \{20\%, 40\%, 60\%, 80\%\}$ and each $N \in \{3, 4, 5\}$:

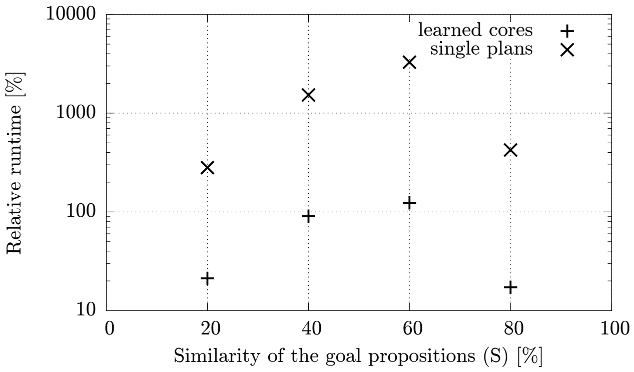


Fig. 2. Relative runtimes corresponding to Steps 4 ('learned cores') and 6 ('single plans') of the experimental protocol. The runtime for Step 5 of the protocol, i.e. for planing from scratch, corresponds to 100% in the figure.

Table 2. Effects of plan cores on the variance of runtimes. The values are standard deviations divided by the mean of the runtimes from Steps 5 and 4 of the protocol, factorized over S.

S	From scratch	Plan-core	S	From scratch	Plan-core
20%	209%	499%	60%	441%	656%
40%	486%	498%	80%	158%	630%

1. We generate the set $\mathcal{P}$ of all planning problems (I_d, G) such that $G \subseteq G_d$ and $\frac{|G|}{|G_d|} \doteq S$. The latter condition states that the goal sets of the two problems are similar to the degree of S.
2. We randomly pick N problems from $\mathcal{P}$ to act as the *training problems*. For each of these problems, the planner generates its optimal plan. These plans constitute the *training set*. We further pick another, *testing problem* from $\mathcal{P}$.
3. *Plan cores* are obtained by generalizing the training plans according to the method described in Section 2. Since the method produces multiple possible joint generalizations, we randomly pick 30 distinct plan cores from the resulting set for evaluation.
4. For each of the 30 plan cores, the planner is launched on the testing problem, starting from a partially constructed plan corresponding to the plan core, and runtime needed to generate a plan is measured, and then averaged over the 30 plan cores.
5. As in Step 4 but the planner starts from scratch rather than from the learned plan core.
6. As in Step 4 but the planner starts from a partially constructed plan corresponding to a single plan randomly selected from the training plans, rather than from the learned plan core.

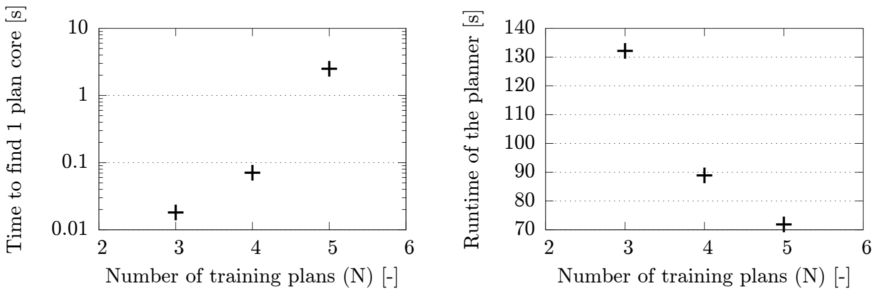


Fig. 3. Average runtimes for learning (top) and planning (bottom) as a function of the number of training plans

Considering Step 1, simple statistical reasoning yields that the smaller the value of S is, the less the training problems generated in Step 2 will be mutually similar, and the less the testing problem will be similar to the training problems.

By comparing results from Steps 4 and 5, we shall assess the influence of our approach to exploiting existing plans, against the situation where existing plans are completely ignored. However, comparing results of Steps 4 and 6 is also important, since Step 6 represents a trivialized variant of our approach, in the sense that it does not require joint generalization of training plans. By this comparison, we thus evaluate the added value brought by the extra theory developed in Section 2.

Results. Fig. 2 shows the resulting runtimes factorized by the 3 strategies corresponding to Steps 4, 5, 6 and the value of S, and averaged over the remaining parameters d and N. Planning procedures not terminating in 1 hour were curtailed and assigned the 1 hour runtime value.

Despite the small slowdown for $S = 60\%$ and the increased variance in runtime, the overall results show that plan cores improved the efficiency of the planner. The overall average result with plan cores achieved 39% runtime of the original. Importantly, results for the scenario where the planner used single training plans instead of generalized plan cores show approximately $10\times$ increase in runtime, which justifies the usage of the learning algorithm.

Of relevance, the absolute time needed for learning is rather negligible in comparison to planning runtimes. This follows from Fig. 3 which plots the runtimes for learning and runtimes for planning using generalized plan cores, factorized by N (i.e., the number of used training plans) and averaged over S and d. Expectedly, the learning times grow and the planning times fall as N increases and its optimal value is, in general, a matter of trade-off.

4 Conclusions

We presented a novel strategy enabling to exploit existing (training) plans in new similar planning tasks by finding a common generalized core of the training plans. In experiments, we showed a substantial speedup of planning resulting from using this strategy. We tested our method in the envisioned application scenario where new planning tasks differ only slightly from the completed tasks used for learning. In particular we assumed that the old and new tasks share the same initial conditions and their goal sets are randomly sampled from the same base. In future work, it would be interesting to refine and test our method also in different instantiations of the similarity assumption, e.g. by allowing that the initial conditions of the new tasks may be different from those of the completed tasks.

Acknowledgment

We gratefully acknowledge the financial support of the Czech Science Foundation (GA ČR) for the years 2008-2010 (grant no. 201/08/0509).

References

1. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: theory and practice. Morgan Kaufmann Publishers, San Francisco (2004)
2. Botea, A., Enzenberger, M., Mller, M., Schaeffer, J.: Macro-FF: improving AI planning with automatically learned macro-operators. Journal of Artificial Intelligence Research 24, 581–621 (2005)
3. Coles, A., Smith, A.: Marvin: a heuristic search planner with online macro-action learning. Journal of Artificial Intelligence Research 28, 119–156 (2007)
4. Chrpa, L., Bartak, R.: Towards getting domain knowledge: Plans analysis through investigation of actions dependencies. In: Florida Artificial Intelligence Conference (2008)
5. Fernandez, S., Aler, R., Borrajo, D.: Using previous experience for learning planning control knowledge. In: Florida Artificial Intelligence Conference (2004)
6. Yoon, S.: Learning heuristic functions from relaxed plans. In: International Conference on Automated Planning and Scheduling. AAAI Press, Menlo Park (2006)
7. Kambhampati, S., Yoon, S.: Explanation based learning for planning. In: Encyclopedia of Machine Learning. Springer, Heidelberg (2010)
8. Dzeroski, S., Raedt, L.D., Blockeel, H.: Relational reinforcement learning. Machine Learning, 7–52 (1998)
9. Huang, Y., Selman, B., Kautz, H.: Learning declarative control rules for constraint-based planning. In: International Conference on Machine Learning (2000)
10. Lorenzo, D., Otero, R.P.: Learning logic programs for action-selection in planning. In: Third International Workshop on Extraction of Knowledge from Databases at the 10th Portuguese Conference on Artificial Intelligence (2001)
11. Plotkin, G., Meltzer, B., Michie, D.: A note on inductive generalization. Machine Intelligence 5, 153–163 (1970)
12. Kuwabara, M., Ogawa, T., Hirata, K., Harao, M.: On generalization and subsumption for ordered clauses. In: New Frontiers in Artificial Intelligence (2006)
13. Tamaddoni-Nezhad, A., Muggleton, S.: The lattice structure and refinement operators for the hypothesis space bounded by a bottom clause. Machine Learning 76, 37–72 (2009)
14. de Raedt, L.: Logical and relational learning. Springer, Heidelberg (2008)

Author Index