Hendrik Blockeel
Jan Ramon
Jude Shavlik
Prasad Tadepalli (Eds.)

LNAI 4894

Inductive Logic Programming

17th International Conference, ILP 2007
Corvallis, OR, USA, June 2007
Revised Selected Papers

Lecture Notes in Artificial Intelligence 4894

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Hendrik Blockeel Jan Ramon
Jude Shavlik Prasad Tadepalli (Eds.)

Inductive Logic Programming

17th International Conference, ILP 2007
Corvallis, OR, USA, June 19-21, 2007
Revised Selected Papers

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Hendrik Blockeel
Jan Ramon
Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A, 3001 Leuven, Belgium
E-mail:{hendrik.blockeel, Jan.Ramon}@cs.kuleuven.be

Jude Shavlik
University of Wisconsin, Madison, WI 53706, USA
E-mail: shavlik@cs.wisc.edu

Prasad Tadepalli
Oregon State University, Corvallis, OR 97331-3202, USA
E-mail: tadepall@cs.orst.edu

Library of Congress Control Number: 2008922294

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

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-540-78468-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-78468-5 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12236039 06/3180 5 4 3 2 1 0

Preface

ILP 2007, the 17th Conference on Inductive Logic Programming, was held in Corvallis, Oregon, USA, June 19–21, and was collocated with the 24th International Conference on Machine Learning. The program consisted of 15 full and 14 short presentations, a poster session, keynote talks by Paolo Frasconi (*Learning with Kernels and Logical Representations*) and David Jensen (*Beyond Prediction: Directions for Probabilistic and Relational Learning*), and several joint sessions with ICML.

Thirty-eight submissions were received this year, out of which fifteen were accepted for publication in the proceedings as full papers and eleven as short papers. Inclusion in the proceedings was decided by taking into account not only the relevance and quality of the work described, but also the quality and level of maturity of the text. Several more submissions were accepted as work-in-progress presentations. Thus the 2007 edition of ILP continued the tradition of adopting high selectivity for published papers, while at the same time offering a forum for work in progress.

All accepted papers were made available in temporary online proceedings during the conference. Revised versions of the submitted papers, incorporating feedback from discussions at the conference, are included either in the proceedings of the conference (this volume) or, for a small number of selected papers, in a special issue of the *Machine Learning* journal (abstracts of these are included in this volume). Papers reporting on work in progress remain available in the online proceedings, at `http://pages.cs.wisc.edu/~shavlik/ilp07wip/`.

The aim of the collocation of ILP and ICML was to promote more interaction between these two communities. The joint sessions with ICML included, besides technical talks from both ILP and ICML, a panel discussion on *Structured Machine Learning: The Next Ten Years*, with panelists Tom Dietterich, Pedro Domingos, Lise Getoor, Stephen Muggleton and Bernhard Pfahringer. ILP participants also had access to the ICML tutorials, several of which were highly relevant to the ILP community. We believe that the large attendance of the joint sessions and the ILP-related tutorials by both communities shows the growing intersection between their interests and confirms the desirability of increased interaction between them.

The ILP 2007 Best Student Paper Award, sponsored by *Machine Learning*, went to Jens Lehmann and Pascal Hitzler for their two papers *Foundations of Refinement Operators for Description Logics* and *A Refinement Operator Based Learning Algorithm for the ALC Description Logic*.

This conference would of course not have been possible without the extensive efforts of the program committee, the additional reviewers, and the student volunteers who helped with the practical organization. We thank all of them for their excellent work. We also gratefully acknowledge the support of Oregon

State University, the *Machine Learning* journal published by Springer, and an anonymous donor. Finally, we thank the International Machine Learning Society for facilitating collaboration.

The next edition of the conference, ILP 2008, will be held in Prague, Czech Republic, chaired by Filip Železný and Nada Lavrač. We wish them a highly successful event.

November 2007

Hendrik Blockeel
Jan Ramon
Jude Shavlik
Prasad Tadepalli

Organization

Program Co-chairs

Hendrik Blockeel (Katholieke Universiteit Leuven, Belgium)
Jude Shavlik (University of Wisconsin, USA)
Prasad Tadepalli (Oregon State University, USA)

Local Arrangements Co-chairs

Prasad Tadepalli (Oregon State University, USA)
Alan Fern (Oregon State University, USA)

Proceedings Chair

Jan Ramon (Katholieke Universiteit Leuven, Belgium)

Program Committee

R. Camacho	Universidade do Porto, Portugal
J. Cussens	University of York, UK
L. Dehaspe	Katholieke Universiteit Leuven, Belgium
L. De Raedt	Katholieke Universiteit Leuven, Belgium
S. Džeroski	Jožef Stefan Institute, Slovenia
F. Esposito	Università di Bari, Italy
A. Fern	Oregon State University, USA
P. Flach	University of Bristol, UK
L. Holder	Washington State University, USA
T. Horvath	University of Bonn and Fraunhofer Institute, Germany
A. Karwath	Albert-Ludwigs-Universität Freiburg, Germany
K. Kersting	Massachusetts Institute of Technology, USA
R. King	University of Wales, Aberystwyth, UK
S. Kramer	Technische Universität München, Germany
N. Lavrač	Jožef Stefan Institute, Slovenia
F. Lisi	Università di Bari, Italy
D. Malerba	Università di Bari, Italy
B. Milch	Massachusetts Institute of Technology, USA
S. Muggleton	Imperial College London, UK
J. Neville	Purdue University, USA

T. Oates	University of Maryland Baltimore County, USA
K. Ohara	Osaka University, Japan
R. Otero	University of Corunna, Spain
D. Page	University of Wisconsin - Madison, USA
B. Pfahringer	University of Waikato, New Zealand
C. Rouveirol	Université Paris-Nord, France
V. Santos Costa	Universidade do Porto, Portugal
M. Sebag	Université Paris-Sud, France
T. Shoudai	Kyushu University, Japan
A. Srinivasan	IBM India Research Laboratory, India
J. Struyf	Katholieke Universiteit Leuven, Belgium
T. Uchida	Hiroshima City University, Japan
C. Vrain	Université d'Orléans, France
S. Wrobel	Fraunhofer Institute for Autonomous Intelligent Systems, Germany
G. Zaverucha	Universidade Federal do Rio de Janeiro, Brazil
F. Železný	Czech Technical University, Czech Republic

Additional Reviewers

André Bergholz
Marenglen Biba
Mario Boley
Maurice Bruynooghe
Tom Croonenborghs
Kurt Driessens
Nicola Fanizzi
Stefano Ferilli
Daan Fierens
Nuno Fonseca
Angelika Kimmig
Chuan Lu
Neville Mehta
Sriraam Natarajan
Aline Paes
Marc Pickett
Scott Proper
Simon Rawles
Jan Ramon
Kate Revoredo
Lisa Torrey
Anneleen Van Assche
Monika Zakova

Webmasters

Louis Oliphant
Heather Rangner
Patrycia Sullivan
Michael Wynkoop

Table of Contents

Invited Talks

Extended Abstracts

Full Papers

Learning with Kernels and Logical Representations

Paolo Frasconi

Dipartimento di Sistemi e Informatica
Università degli Studi di Firenze, Italy
http://www.dsi.unifi.it/~paolo/

Abstract. Choosing an appropriate kernel function is a fundamental step for the application of many popular statistical learning algorithms. Kernels are actually the natural entry point for inserting prior knowledge into the learning process. Inductive logic programming (ILP), on the other hand, offers a powerful and flexible framework for describing existing background knowledge and extracting additional knowledge from the data. It therefore seems natural to explore the synergy between these two important paradigms of machine learning. In this extended abstract (see [1] for a longer version), I briefly review some of our recent work about statistical learning with kernel machines in the ILP setting.

1 Motivations

Statistical and logical approaches to machine learning offer complementary advantages. Logic allows us to represent domain knowledge in a natural and expressive way, and ILP can generate theories and explanations. Statistical learning, on the other hand, allows us to deal with uncertainty and noise in the data. Probabilistic inductive learning programming (PILP), also called statistical relational learning, is a very active area of research and several representational frameworks and models have been proposed during the last few years (see e.g. [2,3] for an overview). It essentially relies on the combined use of logic and probabilities in the learning process.

One interesting distinction that is often made in statistical supervised learning is between generative and discriminant classifiers. In the former case, we typically attempt to model class conditional densities and use Bayes' theorem to obtain the conditional probability of the output label given the input. In the latter case, one attempts to model conditional probabilities directly or, even more simply, to learn a discriminant function that consistently approximates the optimal decision function as the number of training examples grows to infinity. Several PILP approaches are based on generative learning. For example, stochastic logic programs are a generalization of probabilistic context free grammars that assign a probability to each definite clause in a logic program and allow us to infer the probability that a given goal is refuted. The approaches briefly reviewed here take the discriminant direction and exploit classic statistical supervised learning

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 1–3, 2008.

algorithms based on kernel machines. Although several kernels have been defined on discrete data structures like strings, trees, and graphs, there are several motivations for studying the combination of kernels with logic:

- Improving and facilitating kernel design. Background knowledge is usually plugged-in via the kernel function. We can use background knowledge expressed by logic programs and convert it into a kernel, thus embedding it into a statistical learning algorithm in a principled and flexible way.
- Improving the accuracy and the efficiency of existing ILP systems, for example by taking advantage of fast sparse large margin learners such as support vector machines.
- Allowing us to solve in the same framework different learning tasks (e.g. classification, regression, ranking, etc.), which need ad-hoc solutions in the case of typical ILP systems.
- Discovering statistically robust and comprehensible features by learning the kernel function as a logical theory.

2 Overview of Methods

Kernels on Prolog Proof Trees. In this family of kernels (see [4] for details), examples are first mapped into the execution trace of a logic program (called the *visitor*) expressing background knowledge. The similarity between two examples is then computed as the similarity between the corresponding execution traces by using a convolution kernel on Prolog typed ground terms [5]. This method has been applied to artificial and real world problems. It significantly outperforms non-probabilistic ILP algorithms in a classic application to protein fold classification.

Declarative Kernels. In this approach, logic programming allows us to specify a broad class of convolution kernels, providing a simple interface for the incorporation of relational background knowledge. Features are generated by an additional set of mereotopological facts and axioms for reasoning about *parts* and *places*. Declarative kernels have been successfully applied to a large scale text mining task [6]. Compared to state-of-the-art ILP systems, they offer similar or better predictive accuracy and require significantly less computational resources.

kFOIL. In both previous approaches, the background knowledge that defines the kernel function is given in advance. Sometimes, however, human-declared knowledge may be insufficient and we would like to automatically derive a measure of similarity from the available data. kFOIL learns the kernel function by searching a definite clause theory that "covers" the training examples via regularized risk minimization [7]. The kernel in kFOIL is proportional to the number of clauses that are satisfied by the two examples being compared. Clauses are dynamically generated by a greedy search algorithm (in the same spirit as FOIL), guided by the empirical risk associated with the trained kernel machine.

Acknowledgments

I would like to acknowledge the contribution of Luc De Raedt, Niels Landwehr, Stephen Muggleton, and Andrea Passerini to the research summarized in this abstract.

References

1. Frasconi, P., Passerini, A.: Learning with kernels and logical representations. In: De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S. (eds.) Application of Probabilistic Inductive Logic Programming. LNCS (LNAI), vol. 4911. Springer, Heidelberg (2008)
2. De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S.: Application of Probabilistic Inductive Logic Programming. LNCS (LNAI), vol. 4911. Springer, Heidelberg (2008)
3. Getoor, L., Taskar, B.: An Introduction to Statistical Relational Learning, Cambridge, ma edn. MIT Press, Cambridge (2007)
4. Passerini, A., Frasconi, P., De Raedt, L.: Kernels on prolog proof trees: Statistical learning in the ILP setting. Journal of Machine Learning Research 7, 307–342 (2006)
5. Passerini, A., Frasconi, P.: Kernels on prolog ground terms. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, pp. 1626–1627. Edinburgh, Scotland, UK (2005)
6. Frasconi, P., Passerini, A., Muggleton, S., Lodhi, H.: Declarative kernels. In: Kramer, S., Pfahringer, B. (eds.) Inductive Logic Programming. 15th International Conference, ILP 2005, Late-Breaking Papers, pp. 17–19 (2005)
7. Landwehr, N., Passerini, A., Raedt, L.D., Frasconi, P.: kFOIL: Learning simple relational kernels. In: Gil, Y., Mooney, R. (eds.) Proc. Twenty-First National Conference on Artificial Intelligence (AAAI 2006) (2006)

Beyond Prediction: Directions for Probabilistic and Relational Learning

David D. Jensen

Department of Computer Science, University of Massachusetts Amherst,
140 Governors Drive, Amherst, Massachusetts, 01003, USA
jensen@cs.umass.edu

Abstract. Research over the past several decades in learning logical and probabilistic models has greatly increased the range of phenomena that machine learning can address. Recent work has extended these boundaries even further by unifying these two powerful learning frameworks. However, new frontiers await. Current techniques are capable of learning only a subset of the knowledge needed by practitioners in important domains, and further unification of probabilistic and logical learning offers a unique ability to produce the full range of knowledge needed in a wide range of applications.

Keywords: Statistical relational learning, causal inference, quasi-experimental design.

1 Introduction

The past decade has produced substantial progress in unifying methods for learning logical and probabilistic models. We now have a large array of representations, algorithms, performance characterizations, and applications that bring together these two keys to effective learning and reasoning. Much of this work has sought to preserve key attributes of existing learning algorithms (e.g., efficiency) while extending the expressiveness of the representations that can be learned. However, combining probabilistic and logical techniques may open up entirely new capabilities that should not be ignored.

Specifically, these recent advances in machine learning for relational data sets have revealed a surprising new opportunity to learn causal models of complex systems. The opportunity is a potentially deep and unexploited technical interaction between two previously unconnected areas: (1) work in statistical relational learning; and (2) work on quasi-experimental design in the social sciences. Specifically, the type of new data representations conceived and exploited recently by researchers in *statistical relational learning* (SRL) may provide all the information needed to automatically apply powerful statistical techniques from the social sciences known as *quasi-experimental design* (QED). QEDs allow a researcher to exploit unique characteristics of sub-populations of data to make strong inferences about cause-and-effect

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 4 – 21, 2008.

dependencies that would otherwise be undetectable. Such *causal dependencies* infer whether manipulating one variable will affect the value of another variable, and they make such inferences based on non-experimental data.

To date, QEDs have been painstakingly applied by social scientists in an entirely manual way. However, data representations from SRL that record relations (organizational, temporal, spatial, and others) could facilitate automatic application of QEDs. Constructing methods that automatically identify sub-populations of data that meet the requirements of specific QEDs would enable strong and automatic causal inferences from non-experimental data. This fusion of work in SRL and QED would lead to: (1) large increases in the percentage of causal dependencies that can be accurately inferred from non-experimental data; (2) substantial reductions in the amount of data needed to discover causal dependencies that can already be inferred; and (3) reductions in the computational complexity of causal learning algorithms.

If exploited, this capability could substantially improve the ability of researchers to construct causal models of large and complicated systems (e.g., social systems, organizations, and computer systems). Such models would be a significant improvement over existing models learned by statistical and machine learning techniques, the vast majority of which are non-causal (and thus do not allow analysts to correctly infer the effects of potential actions) or only weakly causal (because many of the potential causal dependencies cannot be correctly inferred).

2 Why Causal Models Are Useful

Nearly all algorithms for machine learning analyze data to identify statistical associations among variables. That is, they identify variables of some entity (e.g., a patient's occupation, recent physical contacts, and symptoms) that are statistically associated with other variables (e.g., a disease). Such associations are useful for making predictions about the values of unobserved variables based on the values of variables that can be observed. For example, a doctor could predict whether a patient has a particular disease (an unobserved variable) based on a set of observed symptoms.

Such associational models can be useful in many situations. For example, associational models constructed by machine learning algorithms now sit at the heart of most state-of-the-art systems for machine translation, speech understanding, computer vision, information extraction, and information retrieval. In all of these cases, associations among variables alone are sufficient to meet the goals of the deployed system.

However, machine learning algorithms are often deployed in the hope that they will support decisions about which actions, or *interventions*, to make in a given situation. In the case of medical diagnosis, most medical professionals do not simply want to diagnose disease, but to prevent, treat, or mitigate the effects of the disease as well. They want to know what effect a particular intervention (e.g., implementation of a public health measure or widespread administration of a drug) will have on the health of a population. In such situations, practitioners want models that help them to

design effective interventions, and this requires the modeling of causality, not merely statistical association.

Remarkably, most existing probabilistic models are practically useless for designing effective interventions because they only identify statistical associations, not causal dependencies. As is emphasized in nearly all introductory statistics courses, correlation is not causation — statistical association between two variables does not necessarily imply that one causes the other. For example, suppose we gathered a sample of patients and measured a variety of variables about each patient, including their history of smoking and their incidence of lung cancer [1]. If we analyzed the data, we would very likely find a statistical association between several of these variables.

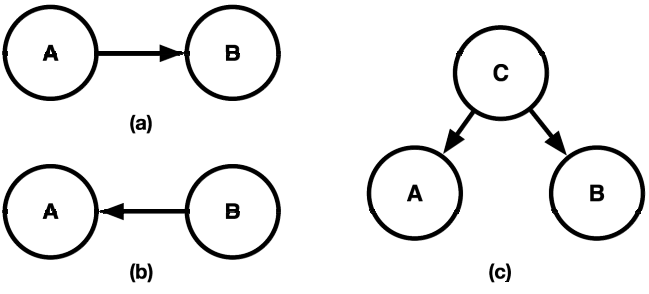


Fig. 1. Three causal models that produce statistical association: *(a) A* causes *B*; *(b) B* causes *A*; and *(c)* a common cause *C* causes both *A* and *B*

However, the association between any two variables *A* and *B* could result from any of three causal situations shown in Figure 1. If *A* and *B* are associated, then *A* could cause *B* (smoking causes lung cancer), *B* could cause *A* (a predisposition to nicotine addiction causes smoking), or a third variable *C* could cause both *A* and *B* (genetics could cause both a predisposition to nicotine addiction and a predisposition to lung cancer)[2]. If a purely associational model is constructed from data and then used to support the design of interventions under the assumption that *A* causes *B*, either of the latter two cases could cause the resulting interventions to be ineffective (or even counterproductive)[3].

[1] This example is inspired by a similar example from Pearl [1].

[2] While the third explanation has been almost entirely discredited through careful experimental and epidemiological analysis, it was at one time a viable causal explanation for the observed association between smoking and lung cancer.

[3] Limited situations involving causal inference among a few variables, such as the case of smoking, genetics, and cancer above, are relatively simple to understand (though not necessarily to analyze). In practice, however, causal models can involve analyzing complex associations among dozens or hundreds of variables. Many such cases have already long surpassed the abilities of human analysts to evaluate without the aid of computational tools, and recent work in relational learning (e.g., [2]) is greatly increasing this complexity by enabling the construction of probabilistic models that analyze associations among variables on vast numbers of inter-related entities. The sections below cover this in substantially more detail.

In contrast, if accurate causal models could be constructed, they would be useful to a wide range of users within science, government, and business. Some of the most obvious uses for causal models are modeling the internal dynamics of organizations and the effects of organizational policies, computer security (e.g., modeling normal and abnormal user behavior in response to system changes and external events), and system design and evaluation (e.g., modeling system behavior to diagnose and improve performance). In addition, many societal goals and functions could be informed by the construction and use of specific causal models, including education and training (e.g., helping to design new educational programs and modify existing ones to maximize effectiveness), logistics (e.g., understanding the effects on the supply chain to improve on-time delivery of goods and services), and healthcare (understanding the causal factors underlying infectious diseases and physical injuries).

3 Example: Fraud Detection at the NASD

One concrete example of the potential opportunities of causal models can be drawn from recent work of the Knowledge Discovery Laboratory (KDL) at the University of Massachusetts Amherst. KDL has been working for several years with the National Association of Securities Dealers (NASD) to analyze large amounts of data on the structure of the US securities industry and the behavior of individuals within that industry ([3], [4], [5]). While the effort to date has produced exclusively associational models, the setting and existing work help make clear the potential applications for causal models.

3.1 NASD

NASD is the primary private-sector regulator of the securities industry in the United States[4]. It is currently responsible for overseeing the activities of more than 5,000 brokerage firms, 170,000 branch offices and 659,000 registered individuals. Since 1939, NASD has worked under the oversight of the U.S. Securities and Exchange Commission (SEC) to regulate all securities firms (called broker-dealers) that conduct business with the public. Currently, NASD employs a staff of over 2,500 employees situated in offices across the country and has an annual operating budget of more than $500 million.

One of NASD's primary aims is to prevent and discover securities fraud and other forms of misconduct by member firms and their employees, called registered representatives, or *reps*. With over 659,000 reps currently employed, it is imperative for NASD to direct its limited regulatory resources towards the parties most likely to engage in risky behavior in the future. It is generally believed by experts at the NASD and others that fraud happens among small, interconnected groups of individuals. Due to the large numbers of potential interactions between reps, timely identification of

[4] NASD has recently changed its name to FINRA (the Financial Industry Regulatory Authority), but we retain the older name here for consistency with earlier papers.

persons and groups of interest is a significant challenge for NASD regulators. The ongoing work is joint effort between researchers in KDL and staff at the NASD to identify effective, automated methods for detecting these high-risk entities to aid NASD in their regulatory efforts.

3.2 Available Data

The primary data employed in this effort is drawn from NASD's Central Registration Depository® (CRD), a collection of records representing all federally registered firms and individuals, including those registered by the SEC, NASD, the states, and other authorized regulators such as the New York Stock Exchange. This depository includes key pieces of regulatory data such as ownership and business location for firms and employment histories for individuals. Although the information in CRD is entirely self-reported, errors, inaccuracies or missing reports can trigger regulatory action by NASD. Since 1981, when the CRD was first created, records on around 3.4 million individuals, 360,000 branches and 25,000 firms have been added to the database.

Some of the most critical pieces of information for NASD's regulatory mission are records of disciplinary actions, typically called disclosures, filed on particular reps. A disclosure can represent any non-compliant actions including regulatory, criminal, or other civil judicial action. In addition, disclosures can also record customer complaints, termination agreements between firms and individual reps, and financial hazards an individual rep experience such as bankruptcies, bond denials, and liens. The disclosure information found in the CRD is one of the primary sources of data on past behavior that NASD uses to assess future risk and focus their regulatory examinations to greatest effect.

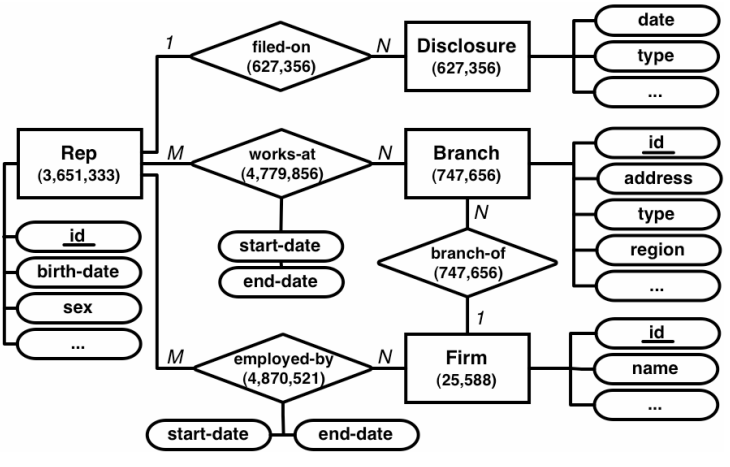


Fig. 2. Entity-relationship diagram for the NASD data

KDL's analysis utilizes data from the CRD about firms, branches, reps and disclosures. The complete entity-relation diagram is shown in Figure 2 along with

counts for each entity appearing in our view of the database. Some of the key aspects of the data set are that it is reasonably large-scale (containing millions of entities), relational and heterogeneous (containing entities and relations of multiple types), and dynamic (representing changes over time in the existence and attributes of entities and relations). In addition, many of the most important statistical associations present in the data are probabilistic.

3.3 The Potential for Causal Models

To date, our analysis of the CRD data has produced several highly useful models that are purely associational. For example, we have constructed statistical models that identify strong associations between the attributes of reps, the branches and firms at which they have worked, and the other reps who work at those organizations ([3],[4]). An extensive double-blind evaluation of these models by NASD analysts showed that they perform well overall and that they perform equivalently to NASD's current expert-derived rules for prioritizing the attention of field examiners [3]. These statistical models could be used in place of, or as an augmentation to, NASD's current methods for identifying high-risk reps and branches.

While highly useful, prioritizing examiner attention affects a relatively small part of NASD's overall regulatory mission. In contrast, causal models could help provide answers to an entirely different class of questions, and they could inform much higher-level decisions. For example, causal models of the securities industry could help inform NASD's approach to these key decisions:

- *Prioritizing new programs* — Given limited resources to devote to a particular region, should NASD place more emphasis on enforcement (e.g., increasing the rate of examinations) or education (e.g., seminars on complying with NASD and SEC rules)?
- *Making closure decisions* — Given that the reps and management of a particular branch have been proven to engage in misconduct, will closing that branch reduce future misconduct (because reps have been deprived of a permissive environment) or increase it (because reps will disperse to other branches and firms, making monitoring more difficult and encouraging questionable behavior at their new organizations)?
- *Responding to economic conditions* — Given a change in a state's economic picture, how are firms in that state likely to respond? How will the changing economic picture interact with different potential NASD's regulatory and education programs in that region to affect the number, composition, and internal regulatory programs of firms in the state?

An accurate answer to each of these questions virtually requires a correct causal model.

4 Current Practice

Given that causal models are so useful, it is not surprising that methods have been developed to learn them from data. At least three classes of statistical techniques have

been developed that differ both in the types of data to which they can be applied and the degree to which they can be applied automatically. One class of techniques, often grouped under the name "experimental design" is useful when analysts have a high degree of control over the situation that produces data. Another class of techniques, which we will call "joint modeling," combines automated methods for estimation of joint probability distributions with a small number of assumptions to infer causality from non-experimental or *observational* data. A final class of techniques, here grouped under the term "quasi-experimental design," seeks to identify situations where observational data meet the assumptions of the techniques from experimental design, allowing causal inferences from observational data.

These three classes of methods face a common set of challenges. First, the techniques must identify whether a statistical association exists between a pair of variables. The basic principles for reliably inferring statistical association — statistical hypothesis testing — have been known for decades, and this step poses relatively little challenge to most manual and algorithmic methods. Second, the techniques must identify the direction of potential causation. This step is most commonly resolved through the use of time (causation is assumed to always run forward in time), but joint modeling can also sometimes resolve this issue in other ways (see below). Finally, the techniques must eliminate the effects of all other potential common causes of the variables, to be certain that the observed association is not just a by-product of other causal influences. As we will see below, each class of methods approaches this final challenge in a unique way.

4.1 Experimental Design

Probably the most commonly used method in the world today for discovering valid causal knowledge is the *controlled randomized experiment*. The rapid expansion of knowledge in the biological, physical, and social sciences over the past 50 years is due in no small part to the available "meta-knowledge" about how to design experiments and analyze their results. Indeed, the discovery, codification, and widespread adoption of methods for experimental design represent one of the major intellectual achievements of the past century.

As the name implies ("controlled randomized experiment"), this set of innovations relies on two key concepts: control and randomization[5]. Control refers generally to the ability of an investigator to intentionally set alternative values of some variable, and to compare the effects of those alternative settings[6]. Control

[5] Here, we omit mention of several other key concepts from experimental design, including replication, orthogonality, and factorial experiments, which are less relevant to the present discussion.

[6] This use of the term "control" is at slight variance with how the term is usually applied in the literature on experimental design. "Control" in this context is often used to refer to a set of subjects that do not receive a treatment (a "control group") as opposed to another group that does receive a treatment. Here, however, we use the term in its broader historical meaning to refer to the ability to set the values of any of a large set of variables so as to keep them constant or to systematically vary their values. This sort of control is so foundational to experimental design that many treatments of the topic omit clear mention of it.

lies at the heart of what is typically meant by an "experiment" and the concept is quite old, stretching back at least as far as John Stuart Mill (1843), and perhaps as far as Hume a century earlier and Bacon a century before that [6]. By controlling variables in an experiment, an investigator can either factor out their effects by holding their value constant or study their effect by systematically varying their values. To do this, however, an investigator must know of the existence of a particular variable and be able to affect its value.

"Randomization" refers to the approach of randomly assigning subjects (e.g., patients in a medical experiment) to treatment groups so that variation in their characteristics that cannot be controlled by the investigator (genetic traits, the effects of unknown medical conditions, etc.) cannot systematically affect the variables being studied. If randomization is employed, the effect of these uncontrolled characteristics will average out across a sufficiently large group. The principles of randomization, and its application to experimental design, was outlined by R.A. Fisher in the 1920s [7], and it has been a staple of experimental design ever since.

What is remarkable about randomization is that it can remove the effect of variables that are *unknown to the investigator*, as long as their values are tied to subjects who can be randomly assigned to treatment groups. For example, an investigator does not have to know which specific genetic factors might influence a patient's response to a particular drug, as long as they randomly assign patients (and thus their genetic factors) to treatment groups.

Investigators who study phenomena in an experimental setting typically control the variables that they can, either systematically altering their values or holding them constant, and randomize most or all the remaining variables. With these two methods, they can study the effects of the variables whose values they can directly manipulate and successfully factor out nearly all other potential causes.

4.2 Joint Modeling

Methods from experimental design assume that you have the ability to control the values of variables and to randomize the assignment of subjects to treatment groups. However, in many cases, investigators do not have the ability to directly manipulate either variable values or subject assignment. Either it is impossible in practice (e.g., it is essentially impossible to set the value of inflation when studying the effect of economic conditions) or such manipulation would be unethical (e.g., though possible in practice, it would be unethical to force subjects to smoke or to work for specific securities firms). Instead, investigators want to draw causal inferences from *observational* data that were gathered without direct manipulation by investigators.[7]

Such observational data sets are increasing in both size and number. They are gathered almost incidentally from a variety of automated systems for accounting,

[7] In some cases, a small degree of experimental control or randomization is available to investigators, but they want to maximize the information-gathering utility of such interventions. In these cases, too, we would like make use of all possible techniques to exploit the observational portions of the data.

auditing, regulation, inventory control, publishing, information retrieval, and communication, and they are often massive in both size and scope. In some cases, the data sets represent the entire population rather than only a sample from it. For example, NASD's CRD database represents essentially every rep operating in the US over several decades.

Investigators have sought to exploit this potential wealth of observational data to infer causal models. Lacking the ability to assure control and randomization, researchers have devised another way to eliminate potential common causes of associated variables: *joint modeling*. This area has been a small but active topic of research for the past several decades in statistics (e.g., [8],[9]), philosophy (e.g., [10]), and computer science (e.g., [1]). The key concept of joint modeling is that, rather than controlling a particular variable, you can estimate the effect of the uncontrolled value and adjust for that effect mathematically. In this way, modeling provides the equivalent of control. If the investigator knows about a potential common cause and can model its effect, then he or she can use that model to factor out the effect of that potential cause.

In all but the simplest cases, this approach requires modeling the joint probability distribution of a set of potential causes, and much of the work in this area has been done using structure learning algorithms for Bayesian networks, which can estimate the joint probability distribution. Using a fairly conventional structure learning algorithm, and making a relatively few additional assumptions, the resulting belief network can be considered to encode causal dependencies[8]. Furthermore, such modeling can be done automatically, once the variables of interest are selected, and the algorithm has the potential to identify all the causal dependencies that hold among that set of variables.

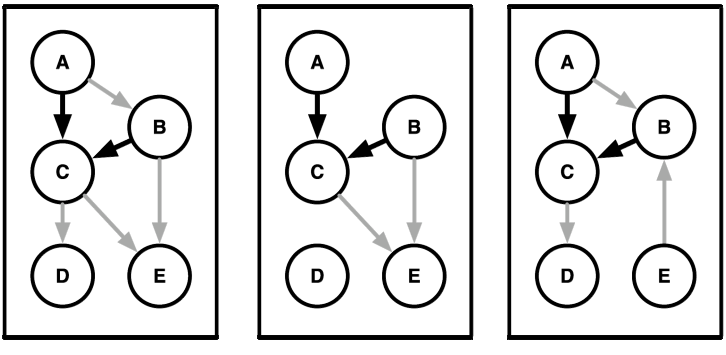


Fig. 3. Three graphical models with partial shared structure

In many cases, structure learning algorithms can identify a set of model structures with equivalent estimated likelihood and which collectively account for

[8] The additional assumptions include *faithfulness*, which specifies that causal effects of different variables do not perfectly cancel out and make it impossible to detect statistical dependence, and *completeness*, which specifies that all potential common causes of any pair of variables included in the model are also included in the model [10].

the vast majority of probability density. If all of these network structures specify that dependence between two variables exists and runs in a single direction, then it is reasonable to identify that dependence as causal. Statistical association exists, runs in a specific direction, and all other potential causes have been accounted for by modeling. For example, Figure 3 shows a simple example with three network structures, each of which shows a somewhat different network structure, but all the structures share a common pair of dependencies showing that *C* depends on *A* and *B*.

4.3 Quasi-experimental Design

A final class of statistical methods is routinely used to support causal inferences. These methods, grouped under the rubric "quasi-experimental design" (QED), attempt to exploit inherent characteristics of observational data sets that partially emulate the control and randomization possible in an experimental setting [11,12][9]. Although QEDs clearly do not always have the internal validity of traditional experimental designs, but they can be applied to the much wider array of data sets that modern data collection practices have made available, and the size and scope of those data sets can partially or completely compensate for the deficiencies that arise from lack of experimental control. Indeed, there are a wide variety of situations where causality can be explored in no other way.

In the absence of explicit control and randomization, some QEDs employ case matching to identify pairs of data instances that are as similar as possible in all respects except for the variable under investigation (the non-equivalent group design). Other QEDs examine how the value of a given variable on the same data instances changes over time, typically before and after some specific event (the regression-discontinuity design). Other types of quasi-experimental designs that have been devised include the proxy pretest design, double pretest design, nonequivalent dependent variables design, pattern matching design, and the regression point displacement design.

A particularly salient example of quasi-experimental design is a classical twin study, a design that has been employed for decades to study the causal factors for particular diseases and conditions. Twin studies compare the incidence of disease in sets of monozygotic (identical) and dizygotic (fraternal) twins. Monozygotic twins share identical genetics, a common fetal environment, and (typically) a common post-natal environment. The same is true for dizygotic twins, except that they are only genetically similar rather than genetically identical.

This remarkable degree of shared background, as well as the specific difference in the shared background between the two types of twins, provides a nearly ideal setting to study the effect of genetics on disease. For example, to identify the degree to which a given condition is due to genetic factors, investigators can determine the correlation in

[9] Here we use a relatively expansive definition of QEDs, including some methods that are not typically covered in textbooks on QED, but which fit well within the general framework of this class of techniques. In particular, we include here methods from hierarchical and multi-level modeling.

the condition among pairs of each type of twin, and then compare the correlation between the two types. A large difference indicates that a large portion of the condition is due to genetics, whereas no difference indicates that the condition is due to other factors. Figure 4 summarizes a few results from twin studies of various conditions.

Fig. 4. Example results from twin studies, drawn from a recent review [13]. Darkest bars indicate effects due to genetics, dark bars to shared environment, and light bars to unique environment.

Two factors are remarkable about the efficacy of twin studies. First, they allow the quantitative impact of genetics to be determined even though investigators may have no idea what specific genes are involved. That is, they can determine the degree to which some variable on a particularly entity (genotype) affects the observed condition without being able to define or measure that specific variable. Second, they can perform this analysis by studying only a tiny fraction of an entire population. Indeed, without access to a very large population, it would be virtually impossible to gain access to a sufficient number of pairs of monozygotic and dizygotic twins. We will return to both of these factors below.

It is also important to note that the validity of twin studies relies on at least three pieces of information known to investigators but not (typically) represented explicitly in data used for QED studies. First, twins occur relatively randomly in the population. If identical twins were much more likely to be born to parents with particular genetic traits or who lived in particular environments, then those factors would confound efforts to use twins to study the effects of genetics on physical conditions. Second, genetic makeup is established temporally prior to the onset of diseases and other conditions. Thus, we know the direction of causality without having to determine it from data.

Finally, and perhaps most importantly, we know that genotype (genetic sequence) and phenotype (physical condition) can be treated as related but separate entities. This means that individuals can have identical genotypes but not identical phenotypes. Thus, the relational representation shown in Figure 5 can be used to represent the data, where monozygotic twins share a common genotype and dizygotic twins do not. This relational representation underlies the inference of investigators that, if the two types of twins do not differ significantly in the correlation of their conditions, then *all possible variables on genotype* can be removed as potential causal factors. Again, we will return to all these three factors in the discussion below.

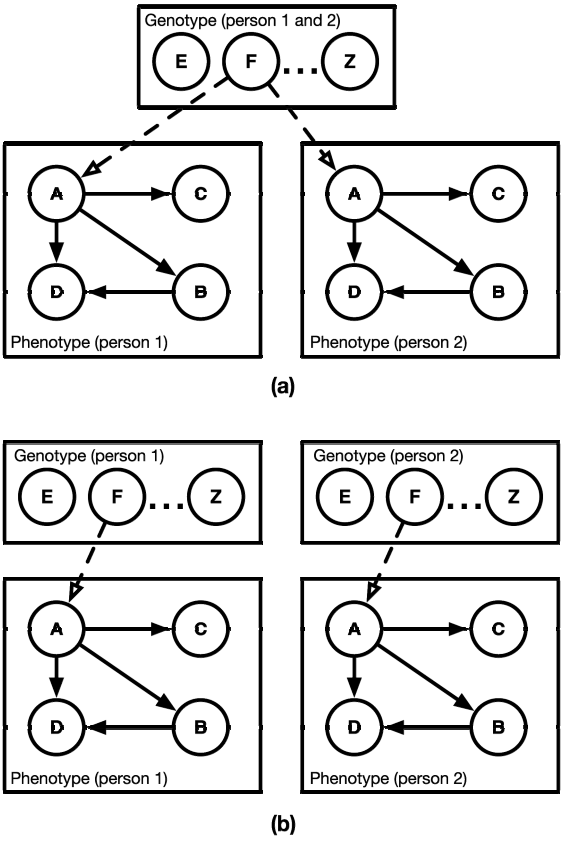


Fig. 5. Graphical models representing monozygotic *(a)* and dizygotic *(b)* twins. Circles represent variables, boxes represent entities, and solid and dashed arrows represent known and potential dependencies, respectively.

5 Limitations of Current Approaches

The availability of three different classes of methods might imply that the existing technical infrastructure for causal inference is sufficient. However, an interacting set of limitations severely restricts the applicability and effectiveness of these methods.

- *Insufficient power* — The only existing methods that are automatic (those from joint modeling) are not capable of discovering all important causal dependencies. The ability of joint inference methods to discover causal dependencies led to a great deal of initial enthusiasm for these methods in the machine learning community. However, as experience has accumulated, it has become clear that these methods often cannot distinguish among a large equivalence class of models that encode the same joint probability distribution. In such cases, joint modeling alone cannot

distinguish among model structures with different edges, cannot resolve the direction of key edges, or both.

- *Computational intractability* — Attempting to learn causal models in non-trivial domains often leads to intractable computational problems due to the need to search large spaces of potential models (vastly increasing the number of models that must be evaluated) and the need to evaluate those models against very large data sets (increasing the effort required to evaluate each model).
- *Manual application* — In contrast to joint modeling methods, quasi-experimental designs must be applied manually. They are not an integral part of automated algorithms for machine learning and knowledge discovery. QEDs are currently used as a way for investigators to manually structure data analysis tasks and to select among different statistical tests, rather than as a component of a machine learning algorithm. This limitation is understandable, given that the applicability of specific QEDs requires knowledge that has not, until quite recently, been explicitly represented in the data provided to machine learning algorithms (relations among entities, temporal information, etc.). However, that situation has changed dramatically over the past decade.
- *Non-relational models* — None of the approaches support construction of joint models of relational domains (domains consisting of interacting entities). The techniques either assume that the domain can be accurately described as a set of independent entities of a single type (in the case of joint modeling), or they do not provide the means to construct a joint probability model (in the case of experimental and quasi-experimental design) [10].

These limitations prevent application of causal modeling to the majority of domains of realistic interest, including the NASD tasks outlined above.

6 Research Opportunities

This raises an intriguing target for future machine learning research: Developing methods for automatically applying quasi-experimental designs to large relational data sets, and to seamlessly integrate these methods for causal inference with methods from joint modeling. We conjecture that the combination of these two classes of methods will produce a large increase in the number of causal dependencies that can be learned, a dramatic decrease in the amount of data needed to learn these dependencies, and a significant decrease in computational complexity of causal inference.

Such research could address three key objectives:

- *Select and enhance data representation and modeling framework* — Researchers in statistical relational learning (SRL) and the larger machine learning community have recently produced a remarkable array of new languages for representing both models and data. Among the representational enhancements are three key types of

[10] It is, in theory, possible for quasi-experimental designs to inform the construction of relational models (indeed, that is one of the primary ideas underlying this proposal). However, this would require something that has not been done to date: combining QEDs with models capable of representing joint probability distributions over relational data.

information needed by QEDs: relations, time, and assignment mechanisms[11]. While at least one existing language can represent each of these types of information, there is not a clear choice among the set of existing languages, and further investigation may reveal additional capabilities necessary to exploit the full range of QEDs. Future work should select, enhance, and (if necessary) develop new data representations and learning frameworks to meet our needs.

- *Develop automated learning methods that apply QEDs* — A large number of specific techniques have been developed in QED and multi-level modeling. Future work should bring those methods into a common framework, develop automated methods for evaluating the applicability and likely validity of specific QEDs, develop querying methods that can select appropriate data samples, and implement algorithms for applying the techniques themselves. These methods will differ substantially from existing machine learning methods, which nearly always apply the same algorithm to the entire data set. In the case of QED, different data will likely be sampled to evaluate each different dependency, and evaluating two different dependencies is likely to require two different designs.
- *Integrate QED and joint modeling methods* — Finally, future work should develop methods to control the application of both QED and traditional joint modeling techniques to integrate these methods into a single powerful system. Such a system will necessitate some moderately complex reasoning because of the interactions among QED methods and joint modeling. For example, joint modeling might be intractable initially, because the number of potential dependencies is so large. However, QED techniques could evaluate specific classes of dependencies and identify whole entities whose variables have no effect on specific variables. They could also identify dependencies that exist with high probability, and thereby vastly decrease the search space for joint modeling. Similarly, the dependencies found by joint modeling may enable or disable specific QEDs.

Once developed, we believe that these methods will lessen or remove the deficiencies noted above. Specifically, the fusion of machine learning and QED that we propose will directly automate application of QEDs and learn relational models. In addition, these methods will indirectly:

- *Increase statistical power* — In the simplest case, QEDs will strictly add to the potential power of a joint modeling approach because they may be able to infer causal dependencies that joint modeling cannot, and they are unlikely to reduce the effectiveness of joint modeling. However, we also expect that they will vastly prune the search space for joint modeling techniques by identifying key causal dependencies and independencies before joint modeling is attempted. This, in turn, will significantly improve the overall power of the combined system.
- *Reduce computational complexity* — Application of QEDs is likely to be a low-complexity operation, both because they can make isolated inferences about specific dependencies without considering all possible dependencies (as joint

[11] "Assignment mechanism" refers to an explicit description of how variables receive their values or two entities become related. For example, one could assume that the occurrence of twins is randomly distributed throughout a population or that the securities industry reps make decisions to join firms based on specific characteristics of the firm as well as their own abilities.

modeling does) and because specific designs exploit very small data sets to make inferences. These inferences, in turn, can prune the search space for joint modeling.

7 Timeliness

This concept of combining work in machine learning and quasi-experimental design appears to be novel. As one indicator, consider the data in Figure 6 showing the results of several queries to Google Scholar, an online database of the full text of technical articles in a wide variety of fields. Approximately 20,000 of the articles mention machine learning or related technologies, and roughly a similar number mention quasi-experimental design. However, only roughly 100 of those articles (0.5%) include one or more terms from both queries. Of those, the vast majority merely mentions one or both technologies rather than making simultaneous use of both.

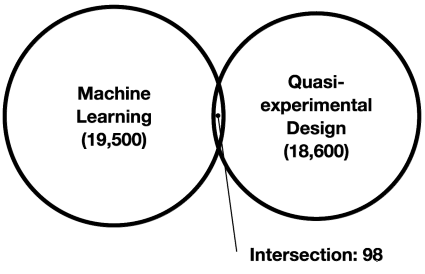


Fig. 6. The relative overlap of articles in machine learning and quasi-experimental design based on hits in Google Scholar with queries of (”machine learning” OR “inductive logic programming” OR “bayesian network” OR “graphical model”) and (”quasi-experimental” OR “quasi experimental”)

Given the apparent value of combining machine learning and QED, why haven't these methods been combined before? First, and most obviously, the methods do not yet exist to automatically apply QEDs, and developing them will be a substantial effort. Second, the majority of recent work in joint modeling has focused on parameter estimation and efficient inference in undirected graphical models, rather than on structure learning in directed models. The former topics are relevant to a variety of relatively low-level processing tasks such as machine translation, speech understanding, computer vision, information extraction, and information retrieval, while the latter topics are the ones relevant to causal inference. Third, automatic application of QEDs virtually requires data representations that encode relations, time, and assignment mechanisms, and such representations have not been available in machine learning until quite recently. Finally, the application of QEDs relies on judicious and highly selective sampling, which runs directly counter to the prevailing practice in machine learning and knowledge discovery of trying to use data sets in their entirety.

While novel, this work builds on at least three key technical developments from the past decade of work in machine learning and knowledge discovery. Specifically:

- *Large observational data sets* — Large observational data sets have become the norm in the field, as well as in related fields such as social network analysis. These data sets provide the scope and size necessary to identify the specific subsets of data that correspond to particular QEDs. For example, in the case of the twin study design mentioned above, a vast initial data set (or at least a population from which data can be drawn) is needed both to identify a sufficient number of pairs of twins and to ensure that a sufficient number of the pairs have at least one member who have a particular disease or condition. Without such vast sets of data, many QEDs could not be effectively applied.
- *Relational data and knowledge representations* — Automatic application of QEDs would be impossible without the sort of rich and expressive data and knowledge representations developed in the past decade[12]. Remarkably, essentially all of the information needed to apply QEDs is encoded in at least one of these representations, although no single representation appears to have every type of information needed.
- *Graph query languages* — Several languages have been developed in the past decade that can extract complex structures from large relational data sets. Unlike traditional query languages (e.g., SQL), these languages return subgraphs (sets of interconnected records, each of which exists in the original graph) rather than single records created for the purpose of summarizing the matching records. This capability is precisely what is needed for checking the applicability of QEDs and carrying out the requisite data sampling.

Given these developments, we believe the time is right to attempt a fusion of machine learning and QED.

8 Risks and Benefits

The risks of this research, while fairly high, are more than compensated for by the large potential benefits of improved techniques for causal inference. Potential risks include:

- *Limited applicability of QEDs* — It could be that QEDs only apply in a relatively narrow range of circumstances. This appears very unlikely, given the vast number of studies in the social sciences that use QEDs.
- *Lack of synergy between QEDs and joint modeling* — It is possible that there is relatively little overlap between the cases where QEDs apply and the cases where joint modeling cannot resolve dependencies.
- *High complexity* — It is possible that effective reasoning about when to apply specific QEDs and joint modeling techniques will have higher complexity than

[12] What is remarkable is not that these richer data representations exist and might be useful, but that methods for joint modeling have been able to infer anything meaningful about causality without them. Every other field of study that attempts to infer causality from observational data implicitly uses such representations, even though the methods are entirely manual.

doing joint inference alone. This seems very unlikely and, even if true, could probably be addressed by using a set of heuristics rather than more precise reasoning methods.

- *Low power due to representational richness* — Increasing the expressiveness of model representations may increase the size of the search space so much that it will overwhelm any gains from applying QEDs. However, all this means is that we have developed imperfect methods for learning models that previously could not be learned at all. In either case, the effort will significantly advance the state of the art in causal inference.

The potential benefits include overcoming the limitations noted in the previous sections, as well as some potential ancillary benefits:

- *General theory of QED* — Quasi-experimental design is currently a loose collection of relatively unrelated designs. It is possible that, in the course of developing automatic methods for applying QEDs, a unified and more general framework for QED will emerge. This would represent not only a computational advance, but a statistical and philosophical one as well.
- *Theory of effective representations* — This work may help identify a formal basis for evaluating the quality of data representations. Specifically, good representations are those that maximize the opportunities for applying QEDs (and thus being able to infer causality). Evaluating representations is a long-standing problem in machine learning, and has been particularly salient in recent work in statistical relational learning.
- *Unifying framework for interdisciplinary research* — If successful, this work would provide a common language for computer scientists, statisticians, philosophers, and social scientists, magnifying the number of researchers working toward common goals and accelerating progress in all of these fields.

Acknowledgments. Discussions with Andrew Fast, Lisa Friedland, Henry Goldberg, Michael Hay, John Komoroske, Marc Maier, Jennifer Neville, Matthew Rattigan, and Agustin Schapira contributed to the ideas in this paper. This material is based on research sponsored by the Air Force Research Laboratory and the Distributive Technology Office (DTO), under agreement number FA8750-07-2-0158. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusion contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory and the Distributive Technology Office (DTO), or the U.S. Government.

References

1. Pearl, J.: Causality: Models, Reasoning, and Inference. Cambridge (2000)
2. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning. MIT Press, Cambridge (2007)
3. Neville, J., Simsek, Ö., Jensen, D., Komoroske, J., Palmer, K., Goldberg, H.: Using Relational Knowledge Discovery To Prevent Securities Fraud. In: Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2005)

4. Fast, A., Friedland, L., Maier, M., Taylor, B., Jensen, D., Goldberg, H., Komoroske, J.: Relational Data Pre-Processing Techniques For Improved Securities Fraud Detection. In: To appear in The Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining (2007)
5. Friedland, L., Jensen, D.: Finding Tribes: Identifying Close-Knit Individuals From Employment Patterns. In: To appear in The Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining (2007)
6. Boring, E.: The Nature and History of Experimental Control. The American Journal of Psychology 67(4), 573–589 (1954)
7. Fisher, R.: Statistical Methods for Research Workers. Oliver and Boyd (1925)
8. Holland, P., Rubin, D.: Causal Inference in Retrospective Studies. Evaluation Review 12, 203–231 (1988)
9. Rubin, D.: Formal Models of Statistical Inference For Causal Effects. Journal of Statistical Planning and Inference 25, 279–292 (1990)
10. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search, 2nd edn. MIT Press, Cambridge (2000)
11. Campbell, D., Stanley, J., Gage, N.: Experimental and Quasi-experimental Designs for Research. Rand McNally (1963)
12. Shadish, W., Cook, T., Campbell, D.: Experimental and Quasi-Experimental Designs. Houghton Mifflin (2002)
13. Boomsma, D., Busjahn, A., Peltonen, L.: Classical Twin Studies and Beyond. Nature Reviews Genetics 3, 872–882 (2002)

Learning Probabilistic Logic Models from Probabilistic Examples (Extended Abstract)

Jianzhong Chen[1], Stephen Muggleton[1], and José Santos[2]

[1] Department of Computing, Imperial College London, London SW7 2AZ, UK
{cjz, shm}@doc.ic.ac.uk
[2] Department of Life Sciences, Imperial College London, London SW7 2AZ, UK
jose.santos06@imperial.ac.uk

This paper describes research in *Probabilistic Inductive Logic Programing* (PILP). The question investigated is whether PILP should always be used to learn from categorical examples. The data sets used by most PILP systems and applications have non-probabilistic class values, like those used in ILP systems. The main reason for this is the lack of an obvious source of probabilistic class values. In this context, we investigate the use of Abductive Stochastic Logic Programs (SLPs) for metabolic network learning.

One of the machine learning approaches, which has been used to model the inhibitory effect of various toxins in the metabolic network of rats, is abductive ILP [3]. A group of rats are injected with a toxin and the changes on the concentrations of a number of chemical compounds are monitored over time. The binary information on up/down regulations of metabolite concentrations is combined with background knowledge representing a subset of the KEGG metabolic diagrams. An abductive ILP program is used to suggest the inhibitory effects occurring in the network.

Abductive SLPs [1] are a learning framework that supports abduction in SLPs to provide a probability distribution over the abductive hypotheses based on their *possible worlds*. An *abductive SLP* S_A is a first order Stochastic Logic Program that supports stochastic abduction. To learn an S_A, we assume a background knowledge theory B and a set of independently observed ground probabilistic examples E. The learning constructs a set of labelled hypothesised abducibles $H = \{p : ha\}$ such that when added to S_A, we have $B \wedge H \models E$ and the labels $\{p\}$ are chosen to maximize the likelihood of H given E and B. We perform SLP parameter estimation algorithms, using FAM [2], to learn the probabilities for a given set of abducibles.

We have a scientific data set collected from some control cases as well as a set of data points from treated cases. All the data are mutually independent. Table 1 presents an algorithm applied to our rat metabolic network inhibition data set for extracting the probabilistic examples from empirical data containing control and treated cases.

The experiments include two tasks – learning abductive SLP_C from categorical examples and learning abductive SLP_P from probabilistic examples. The null hypotheses is that the predictive accuracy of an SLP_P model **does not** outperform an SLP_C model for predicting the concentration level of metabolites in a given rat metabolic network inhibition experiment. The empirical proba-

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 22–23, 2008.

Table 1. Algorithm of estimating empirical probabilities

1. Initialize a matrix MR with column=2 and row=number of metabolites;
2. for each metabolite α do:
2.1. C_α=$\{concentration(\alpha)\}$, a set of α values observed in the **control** cases;
2.2. M_α=MEAN(C_α),SD_α=STANDARDDEVIATION(C_α);
2.3. T_α=$\{concentration'(\alpha)\}$, a set of τ_α values observed in the **treatment** cases;
2.4. $MR[\alpha, 1] = M_\alpha <$ MEAN(T_α) ? Up : Down;
2.5. $MR[\alpha, 2]$= MEAN($\{$PNORM($\tau_\alpha, M_\alpha, SD_\alpha$)$\}$);
3. Apply matrix MR in the abductive SLP learning

bilities are extracted from the raw data consisting of the concentration level of 20 metabolites on 20 rats (10 control cases and 10 treated cases) after 8 hours of the injection of hydrazine. We apply a leave-one-out cross validation process to do the prediction. The evaluation of the prediction models is made by calculating the predictive accuracy of SLP_C and SLP_P against the estimated empirical probabilities respectively. In particular, when evaluating only with the categorical observations, SLP_C and SLP_P correctly predicted 9 and 11 out of 20 metabolites respectively, while the ILP model has 11 correct predictions; when evaluating with the probabilistic examples, SLP_P outperforms SLP_C by 72.74% against 68.31% in average predictive accuracy (with a significance level of 0.041). By comparing the learned SLP model with the previous ILP model, at least two promising new findings have been discovered in the SLP model, which can be explained by the introduced empirical probabilities. In addition, the SLP models learned not only the patterns but also the degree of belief of the patterns which improve the insight from the learned models.

In this study, we revisit an application developed originally using ILP by replacing the underlying logic program description with SLPs, one of the underlying PILP frameworks. In both the ILP and PILP cases a mixture of abduction and induction are used. In conclusion, the null hypotheses were rejected on the bases of the theoretical and experimental results. Our results demonstrate that the PILP approach not only leads to a significant decrease in error accompanied by improved insight from the learned result but also provides a way of learning probabilistic logic models from probabilistic examples.

References

1. Arvanitis, A., Muggleton, S.H., Chen, J., Watanabe, H.: Abduction with stochastic logic programs based on a possible worlds semantics. In: Short Paper Proceedings of the 16th International Conference on Inductive Logic Programming, University of Corunna (2006)
2. Cussens, J.: Parameter estimation in stochastic logic programs. Machine Learning 44(3), 245–271 (2001)
3. 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)

Learning Directed Probabilistic Logical Models Using Ordering-Search

Daan Fierens, Jan Ramon, Maurice Bruynooghe, and Hendrik Blockeel

K.U.Leuven, Dept. of Computer Science, Celestijnenlaan 200A, 3001 Heverlee, Belgium
{Daan.Fierens, Jan.Ramon, Maurice.Bruynooghe,
Hendrik.Blockeel}@cs.kuleuven.be

There is an increasing interest in upgrading Bayesian networks to the relational case, resulting in directed probabilistic logical models. Many formalisms to describe such models have been introduced and learning algorithms have been developed for several such formalisms. Most of these algorithms are upgrades of the traditional *structure-search* algorithm for Bayesian networks. However, in 2005 an alternative algorithm for learning Bayesian networks, *ordering-search*, was introduced that performs at least as well as structure-search while usually being faster. This motivated us to develop an ordering-search algorithm for learning directed probabilistic logical models.

Our ordering-search algorithm is based on the observation that learning a model is relatively easy when an ordering on the predicates is given and each predicate has as potential parents only the predicates that precede it in the ordering (this implies that only non-recursive models are learned). Given such an ordering, we can learn for each predicate separately which of its potential parents are the effective parents. This can simply be done by learning a logical probability tree for that predicate with as inputs all potential parents. The effective parents are then determined as all the predicates that are effectively used in the learned tree. Since often no ordering on the predicates is known beforehand, ordering-search performs hillclimbing through the space of orderings to determine the optimal ordering, in each step applying the above procedure.

We implemented the above ordering-search algorithm for the formalism Logical Bayesian Networks. We also implemented a structure-search algorithm that is very similar to the existing structure-search algorithms for related formalisms. We experimentally compared both algorithms in terms of quality (likelihood of test data) and compactness of the learned models and computational efficiency. We performed experiments on two relational domains: a synthetic domain for which we generated datasets of varying size by sampling from a given model and the UWCSE dataset.

Our experimental results show that ordering-search is competitive with structure-search in terms of quality and compactness of the learned models. However, ordering-search is significantly faster. We conclude that ordering-search is a good alternative to structure-search for learning directed probabilistic logical models.

References

1. Fierens, D., Ramon, J., Bruynooghe, M., Blockeel, H.: Learning Directed Probabilistic Logical Models: Ordering-Search versus Structure-Search. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) ECML 2007. LNCS (LNAI), vol. 4701, Springer, Heidelberg (2007)

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, p. 24, 2008.

Learning to Assign Degrees of Belief in Relational Domains

Frdric Koriche

LIRMM, Universit Montpellier II
161 Rue Ada, 34392 Montpellier Cedex 5, France
Frederic.Koriche@lirmm.fr

As uncertainty pervades the real world, it seems obvious that the decisions we make, the conclusions we reach, and the explanations we offer are usually based on our judgements of the probability of uncertain events such as success in a new medical treatment or the state of the market. For example, if an agent wishes to employ the expected-utility paradigm of decision theory in order to guide its actions, it must assign subjective probabilities to various assertions. Less obvious, however, is the question of *how* to elicit such degrees of beliefs.

The standard knowledge representation approach claims that the agent starts its life-cycle by acquiring a pool of knowledge expressing several constraints about its environment, such as properties of objects and relationships among them. This information is stored in some knowledge base using a logical representation language [1,4,8] or a graphical representation language [3,9,11]. After this period of knowledge preparation, the agent is expected to achieve optimal performance by evaluating any query with perfect accuracy. Indeed, according to the well-defined semantics of the representation language, a knowledge base provides a compact representation of a probability measure that can be used to evaluate queries. For example, if we select first-order logic as our representation language, the probability measure is induced by assigning equal likelihood to all models of the knowledge base; the degree of belief of any given query is thus the fraction of those models which are consistent with the query.

From a pragmatic perspective, the usefulness of a computational framework for assigning subjective probabilities depends both on the *accuracy* of the belief estimates and the *efficiency* of belief estimation. Unfortunately, in the standard knowledge representation approach, the task of assigning subjective probabilities can very much demanding from a computational point of view. In contrast, the *learning to reason* (L2R) framework has recently emerged as an active research field of ILP for dealing with the intractability of reasoning problems [5,6,12]. By incorporating a role of inductive learning within reasoning, this approach stresses the importance of combining the processes of knowledge acquisition and query evaluation together. The main departure from the classical approach is that knowledge is not ascribed *a priori*, in the purpose of describing an environment, but instead acquired *a posteriori*, by experience, in order to improve the agent's ability to reason efficiently in its environment.

Following the L2R paradigm, this study aims at eliciting degrees of beliefs in an inductive manner, using a computational model of learning. Namely, the world, or the domain in question, is modeled as a probability distribution W on

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 25–26, 2008.

a space of relational interpretations. To acquire knowledge from the world, the agent is given a "grace period" in which it can interact with its learning interface. The purpose of the learning interface is to help the agent in concentrating its effort toward finding a representation KB of W that is useful for evaluating queries in some target query language $\mathcal{Q}$. The reasoning performance is measured only after this period, when the agent is presented with new queries from $\mathcal{Q}$ and has to estimate their probability according to its representation KB.

We develop an online L2R algorithm which combines techniques in regression learning and weighted model counting. The algorithm uses an exponentiated gradient strategy [7] adapted for assigning probabilities to relational queries. The total number of mistakes made by the reasoner depends only *logarithmically* in the size of the target probability distribution, and hence *linearly* in the input dimension. Thus, the learning curve of the reasoner is guaranteed to converge to yield accurate estimations after a polynomial number of interactions. The key idea behind efficient query evaluation lies in a representation of the "mistake-driven" knowledge that allows tractable forms of weighted model counting [10]. Namely, for various fragments of the relational language $\mathcal{R}$ in [2], the computational cost of assigning degrees of belief is polynomial in the number of mistakes made so far, and hence, the input dimension. This result highlights the interest of the L2R framework by providing efficient solutions to relational probabilistic reasoning problems that are provably intractable in the classical framework.

References

1. Bacchus, F., Grove, A.J., Halpern, J.Y., Koller, D.: From statistical knowledge bases to degrees of belief. Artificial Intelligence 87(1-2), 75–143 (1996)
2. Cumby, C.M., Roth, D.: Relational representations that facilitate learning. In: 17th Int. Conf. on Knowledge Representation and Reasoning, pp. 425–434 (2000)
3. Jaeger, M.: Relational bayesian networks. In: Proc. of the 13th Conference on Uncertainty in Artificial Intelligence, Providence, RI, pp. 266–273. Morgan Kaufmann, San Francisco (1997)
4. Kersting, K., De Raedt, L.: Adaptive bayesian logic programs. In: 11th Int. Conference on Inductive Logic Programming, pp. 104–117 (2001)
5. Khardon, R., Roth, D.: Learning to reason. ACM Journal 44(5), 697–725 (1997)
6. Khardon, R., Roth, D.: Learning to reason with a restricted view. Machine Learning 35(2), 95–116 (1999)
7. Kivinen, J., Warmuth, M.K.: Exponentiated gradient versus gradient descent for linear predictors. Information and Computation 132(1), 1–63 (1997)
8. Poole, D.: Probabilistic horn abduction and Bayesain networks. Artificial Intelligence 64, 81–129 (1993)
9. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62(1-2), 107–136 (2006)
10. Sang, T., Beame, P., Kautz, H.A.: Performing Bayesian inference by weighted model counting. In: 20h National Conference on Artificial Intelligence (AAAI), pp. 475–482 (2005)
11. Taskar, B., Abbeel, P., Koller, D.: Discriminative probabilistic models for relational data. In: Proc. of the 18th Conference in Uncertainty in Artificial Intelligence, Edmonton, Alberta, Canada, pp. 485–492. Morgan Kaufmann, San Francisco (2002)
12. Valiant, L.G.: Robust logics. Artificial Intelligence 117(2), 231–253 (2000)

Bias/Variance Analysis for Relational Domains

Jennifer Neville[1] and David Jensen[2]

[1] Departments of Computer Science and Statistics, Purdue University

[2] Department of Computer Science, University of Massachusetts Amherst

Extended Abstract

Bias/variance analysis [1] is a useful tool for investigating the performance of machine learning algorithms. Conventional analysis decomposes loss into errors due to aspects of the learning process with an underlying assumption that there is no variation in model predictions due to the inference process used for prediction. This assumption is often violated when *collective inference* models are used for classification of relational data.

In relational data, when there are dependencies among the class labels of related instances, the inferences about one object can be used to improve the inferences about other related objects. Collective inference techniques exploit these dependencies by *jointly* inferring the class labels in a test set. This approach can produce more accurate predictions than conditional inference for each instance independently, but it also introduces an additional source of error, both through the use of approximate inference algorithms and through variation in the availability of test set information. To date, the impact of *inference* error on relational model performance has not been investigated.

We propose a new bias/variance framework[1] that decomposes marginal squared-loss error into components of both the *learning* process, used to estimate the model, and the *inference* process, used for prediction. To illustrate the decomposition, consider the following procedure. We measure the variation of model predictions for an instance x in two ways. First, when we generate synthetic data we record the data generation probability as the optimal prediction for x as y^*. Next, we record marginal predictions for x from models learned on different training sets, allowing the optimal predictions of related instances ($\mathbf{y}^*_{\mathbf{R}}$) to be used during inference. These predictions form the *learning* distribution, with a mean *learning* prediction for x of y_{Lm}. Finally, we record predictions for x from models learned on different training sets, where each learned model is applied for prediction a number of times on the test set. These predictions form the *total* distribution, with a mean *total* prediction for x of y_{Tm}. The model's *learning* bias for x is $B_L(x) = (y^* - y_{Lm})^2$; the *inference* bias is $B_I(x) = (y_{Lm} - y_{Tm})^2$. The model's *learning* variance is $V_L(x) = E[(y_{Lm} - y_L)^2]$; the total variance is defined analogously $V_T(x) = E[(y_{Tm} - y)^2]$; the *inference* variance is the difference between the *total* variance and the *learning* variance $V_I(x) = V_T(x) - V_L(x)$.

[1] We refer the reader to the longer version of this paper [4] for a complete discussion of the decomposition, methodology, and experimental results.

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 27–28, 2008.

We evaluate relational model performance within this framework, using both synthetic and real-world datasets, by measuring squared loss and decomposing it into bias and variance components for each model. We compare the performance of three models: relational Markov networks (RMNs) [5], relational dependency networks (RDNs) [2], and latent group models (LGMs) [3].

The experiments measure model performance over a range of data characteristics and show that (1) inference can be a significant source of error, and (2) the models exhibit different types of errors as data characteristics are varied. In particular, graph structure, autocorrelation dependencies, and amount of test set labeling, all affect relative relational model performance. LGMs are more robust to sparse labeling and perform well when the underlying data exhibit high clustering. When the underlying clustering is low, LGMs experience high learning bias due to poor group identification. RDNs, applied with Gibbs sampling, experience high variance on test data with sparse labeling, but perform well across a wide range of graph structures. RMNs, applied with loopy belief propagation, have higher bias on densely connected graphs, but are more robust to sparse test set labeling.

The analysis suggests a number of directions to pursue to improve model performance—either by incorporating properties of the inference process into learning or through modification of the inference process based on properties of learning. To improve LGM performance, we need to improve the initial identification of clusters. This may be achieved through the development of a joint learning procedure that clusters for groups while simultaneously estimating attribute dependencies in the model. To improve RDN performance, we need to improve inference when there are few labeled instances in the test set. This may be achieved by using meta-knowledge about the test set to bias the feature selection process during learning. Finally, to improve RMN performance, we need to improve inference accuracy when the underlying graph connectivity is high. This may be achieved through the use of approximate inference techniques other than loopy belief propagation, or through the use of aggregate features in clique templates (that summarize cluster information) rather than using redundant pairwise features.

References

1. Geman, S., Bienenstock, E., Doursat, R.: Neural networks and the bias/variance dilemma. Neural Computation 4, 1–58 (1992)
2. Neville, J., Jensen, D.: Dependency networks for relational data. In: Proceedings of the 4th IEEE International Conference on Data Mining, pp. 170–177 (2004)
3. Neville, J., Jensen, D.: Leveraging relational autocorrelation with latent group models. In: Proceedings of the 5th IEEE International Conference on Data Mining, pp. 322–329 (2005)
4. Neville, J., Jensen, D.: A bias-variance decomposition for collective inference models. Machine Learning Journal, under submission (invited)
5. Taskar, B., Abbeel, P., Koller, D.: Discriminative probabilistic models for relational data. In: Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence, pp. 485–492 (2002)

Induction of Optimal Semantic Semi-distances for Clausal Knowledge Bases

Claudia d'Amato, Nicola Fanizzi, and Floriana Esposito

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

Abstract. Several activities related to semantically annotated resources can be enabled by a notion of similarity, spanning from clustering to retrieval, matchmaking and other forms of inductive reasoning. We propose the definition of a family of semi-distances over the set of objects in a knowledge base which can be used in these activities. In the line of works on distance-induction on clausal spaces, the family is parameterized on a committee of concepts expressed with clauses. Hence, we also present a method based on the idea of simulated annealing to be used to optimize the choice of the best concept committee.

1 Introduction

Assessing semantic similarity between objects can support a wide variety of instance-based tasks spanning from *case-based reasoning* and *retrieval* to *inductive generalization* and *clustering*.

As pointed out in related surveys [16], initially, most of the proposed similarity measures for concept descriptions focus on the similarity of atomic concepts within simple concept hierarchies or are strongly based on the structure of the terms for specific FOL fragments [5]. Alternative approaches are based on related notions of *feature* similarity or *information content*. All these approaches have been specifically aimed at assessing similarity between concepts (see also [10]). In the perspective of exploiting similarity measures in inductive (instance-based) tasks like those mentioned above, the need for a definition of a semantic similarity measure for *instances* arises [1, 2, 12].

Recently, semantic dissimilarity measures for specific FOL fragments have been proposed which turned out to be practically effective for the targeted inductive tasks. Although these measures ultimately rely on the semantics of primitive concepts as elicited from the knowledge base, still they are partly based on structural criteria (a notion of normal form) which determine also their main weakness: they are hardly portable to deal with other FOL fragments.

Therefore, we have devised a new family of dissimilarity measures for semantically annotated resources, which can overcome the aforementioned limitations. Our measures are mainly based on Minkowski's measures for Euclidean spaces defined by means of the *hypothesis-driven* distance induction method [14]. Another source of inspiration was provided by the *indiscernibility* relationships investigated *rough sets* theory [11].

Namely, the proposed measures are based on the degree of discernibility of the input objects with respect to a committee of features, which are represented by concept

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 29–38, 2008.

descriptions. As such, these new measures are not absolute, since they depend on both the choice (and cardinality) of the features committee and the knowledge base they are applied to. Rather, they rely on statistics on objects that are likely to be maintained by the knowledge base management system, which can determine a potential speed-up in the measure computation during knowledge-intensive tasks. Differently from the original idea [14], we give a definition of the notion of projections which is based on model-theory in LP.

Furthermore, we also propose ways to extend the presented measures to the case of assessing concept similarity by considering concepts as represented by their extension, i.e. the set of their instances. Specifically, we recur to notions borrowed from clustering [6] such as the *medoid*, the most centrally located instance in a concept extension w.r.t. a given metric.

Experimentally[1], it may be shown that the measures induced by large committees (e.g. including all primitive and defined concepts) can be sufficiently accurate when employed for classification tasks even though the employed committee of features were not the optimal one or if the concepts therein were partially redundant. Nevertheless, this has led us to investigate on a method to optimize the committee of features that serve as dimensions for the computation of the measure. To this purpose, the employment of genetic programming and randomized search procedures was considered. Finally we opted for an optimization search procedure based on *simulated annealing* [7], a randomized approach that can overcome the problem of the search being caught in local minima.

The remainder of the paper is organized as follows. The definition of the family of measures is proposed in Sect. 2, where we prove them to be semi-distances and extend their applicability to the case of concept similarity. In Sect. 3, we illustrate and discuss the method for optimizing the choice of concepts for the committee of features which induces the measures. The effectiveness of the method is demonstrated in a preliminary experimentation (see Sect. 4) on the task of similarity search. Possible developments are finally examined in Sect. 5.

2 A Family of Semi-distances for Instances

In the following, we assume that objects (instances), concepts and relationships among them may be defined in terms of a function-free (yet not constant-free) clausal language such as DATALOG, endowed with the standard semantics (see [9] for reference).

We will consider a *knowledge base* $\mathcal{K} = \langle \mathcal{P}, \mathcal{D} \rangle$, where $\mathcal{P}$ is a logic program representing the *schema*, with concepts (entities) and relationships defined through definite clauses, and the *database* $\mathcal{D}$ is a set of ground facts concerning the world state. In this context, without loss of generality, we will consider concepts as described by unary atoms. *Primitive* concepts are defined in $\mathcal{D}$ extensionally by means of ground facts only, whereas *defined* concepts will be defined in $\mathcal{P}$ by means of clauses. The set of the objects occurring in $\mathcal{K}$ is denoted with $\mathsf{const}(\mathcal{D})$.

[1] Such experiments, regarding a nearest neighbor search task, are not further commented here for the sake of brevity.

As regards the necessary inference services, our measures will require performing *instance-checking*, which amounts to determining whether an object belongs (is an instance) of a concept in a certain interpretation.

2.1 Basic Measure Definition

It can be observed that instances lack a syntactic structure that may be exploited for a comparison. However, on a semantic level, similar objects should *behave* similarly with respect to the same concepts, i.e. similar assertions (facts) should be shared. Conversely, dissimilar instances should likely instantiate disjoint concepts.

Therefore, we introduce novel dissimilarity measures for objects, whose rationale is the comparison of their semantics w.r.t. a fixed number of dimensions represented by concept descriptions (predicate definitions). Namely, instances are compared on the grounds of their behavior w.r.t. a reduced (yet not necessarily disjoint) committee of features, represented by a collection of concept descriptions, say $\mathsf{F} = \{F_1, F_2, \ldots, F_m\}$, which stands as a group of discriminating *features* expressed in the language taken into account. In this case, we will consider unary predicates which have a definition in the knowledge base.

Following [14], a family of totally semantic distance measures for objects can be defined for clausal representations. In its simplest formulation, inspired by Minkowski's metrics, these functions can be defined as follows:

Definition 2.1 (family of measures). *Let $\mathcal{K}$ be a knowledge base. Given a set of concept descriptions $\mathsf{F} = \{F_1, F_2, \ldots, F_m\}$, a family $\{d_p^{\mathsf{F}}\}_{p \in \mathbb{N}}$ of functions $d_p^{\mathsf{F}} : \mathsf{const}(\mathcal{D}) \times \mathsf{const}(\mathcal{D}) \mapsto [0, 1]$ is defined as follows:*

$$\forall a, b \in \mathsf{const}(\mathcal{D}) \quad d_p^{\mathsf{F}}(a, b) := \frac{1}{m} \left[\sum_{i=1}^{m} \mid \pi_i(a) - \pi_i(b) \mid^p \right]^{1/p}$$

*where $\forall i \in \{1, \ldots, m\}$ the i-*th projection function *π_i is defined by:*

$$\forall a \in \mathsf{const}(\mathcal{D}) \quad \pi_i(a) = \begin{cases} 1 & \mathcal{K} \vdash F_i(a) \\ 0 & \textit{otherwise} \end{cases}$$

The superscript F will be omitted when the set of features is fixed.

2.2 Discussion

We can prove that these functions have the standard properties for semi-distances:

Proposition 2.1 (semi-distance). *For a fixed feature set and $p \in \mathbb{N}$, function d_p is a semi-distance.*

Proof. In order to prove the thesis, given any three objects $a, b, c \in \mathsf{const}(\mathcal{D})$ it must hold that:

1. $d_p(a, b) \geq 0$ *positivity*
2. $d_p(a, b) = d_p(b, a)$ *symmetry*
3. $d_p(a, c) \leq d_p(a, b) + d_p(b, c)$ *triangular inequality*

Now, we observe that:

1. *trivial, by definition*
2. *trivial, for the commutativity of the operators involved*
3. *it follows from the properties of the power function:*

$$
\begin{aligned}
d_p(a,c) &= \frac{1}{m}\left[\sum_{i=1}^{m} \mid \pi_i(a) - \pi_i(c) \mid^p\right]^{1/p} \\
&= \frac{1}{m}\left[\sum_{i=1}^{m} \mid \pi_i(a) - \pi_i(b) + \pi_i(b) - \pi_i(c) \mid^p\right]^{1/p} \\
&\leq \frac{1}{m}\left[\sum_{i=1}^{m} \mid \pi_i(a) - \pi_i(b) \mid^p + \mid \pi_i(b) - \pi_i(c) \mid^p\right]^{1/p} \\
&= \frac{1}{m}\left[\sum_{i=1}^{m} \mid \pi_i(a) - \pi_i(b) \mid^p + \sum_{i=1}^{m} \mid \pi_i(b) - \pi_i(c) \mid^p\right]^{1/p} \\
&\leq \frac{1}{m}\left[\sum_{i=1}^{m} \mid \pi_i(a) - \pi_i(b) \mid^p\right]^{1/p} + \frac{1}{m}\left[\sum_{i=1}^{m} \mid \pi_i(b) - \pi_i(c) \mid^p\right]^{1/p} \\
&= d_p(a,b) + d_p(b,c)
\end{aligned}
$$

As such, these are only a semi-distances. Namely, it cannot be proved that $d_p(a,b) = 0$ iff $a = b$. This is the case of *indiscernible* instances with respect to the given set of features F [11].

Here, we make the assumption that the feature-set F may represent a sufficient number of (possibly redundant) features that are able to discriminate really different objects. As hinted in [14], redundancy may help appreciate the relative differences in similarity.

Compared to other proposed distance (or dissimilarity) measures, the presented functions are not based on structural (syntactical) criteria; namely, they require only deciding whether an object can be an instance of the concepts in the committee.

Note that the computation of projection functions can be performed in advance (with the support of suitable DBMSs) thus determining a speed-up in the actual computation of the distance measure. This is very important for the integration of these measures in instance-based methods which massively use distances, such as in case-based reasoning and clustering.

2.3 Extensions

The definition above might be further refined and extended by recurring to model theory. Namely, the set of Herbrand models of the knowledge base $\mathcal{M}_\mathcal{K} \subseteq 2^{|\mathcal{B}_\mathcal{K}|}$ may be considered, where $\mathcal{B}_\mathcal{K}$ stands for its Herbrand base.

Now, given two instances a and b to be compared w.r.t. a certain feature F_i, $i = 1, \ldots, m$, we might check whether they can be distinguished in the world represented by a Herbrand interpretation $\mathcal{I} \in \mathcal{M}_\mathcal{K}$: $\mathcal{I} \models F_i(a)$ and $\mathcal{I} \models F_i(b)$. Hence, a distance measure should count the cases of disagreement, varying the Herbrand models of the knowledge base: The resulting definition for a dissimilarity measure is the following:

$$\forall a, b \in \mathsf{const}(\mathcal{D}) \quad d_p^{\mathsf{F}}(a,b) := \frac{1}{m \cdot |\mathcal{M}_{\mathcal{K}}|} \left[\sum_{\mathcal{I} \in \mathcal{M}_{\mathcal{K}}} \sum_{i=1}^{m} \mid \pi_i^{\mathcal{I}}(a) - \pi_i^{\mathcal{I}}(b) \mid^p \right]^{1/p}$$

where the projections are computed for a specific world state as encoded by a Herbrand interpretation $\mathcal{I}$:

$$\forall a \in \mathsf{const}(\mathcal{D}) \quad \pi_i^I(a) = \begin{cases} 1 & F_i(a) \in \mathcal{I} \\ 0 & \textit{otherwise} \end{cases}$$

Following the rationale of the average link criterion used in clustering [6], the measures can be extended to the case of concepts, by recurring to the notion of medoids. The *medoid* of a group of objects is the object that has the highest similarity w.r.t. the others. Formally. given a group $G = \{a_1, a_2, \ldots, a_n\}$, the medoid is defined:

$$m = \text{medoid}(G) = \operatorname*{argmin}_{a \in G} \sum_{j=1}^{n} d_p^{\mathsf{F}}(a, a_j)$$

Now, given two concepts C_1, C_2, we can consider the two corresponding groups of objects obtained by retrieval $R_i = \{a \in \mathsf{const}(\mathcal{D}) \mid \mathcal{K} \models C_i(a)\}$, and their resp. medoids $m_i = \text{medoid}(R_i)$ for $i = 1, 2$ w.r.t. a given measure d_p^{F} (for some $p > 0$ and committee F). Then we can define the function for concepts as follows:

$$d_p^{\mathsf{F}}(C_1, C_2) := d_p^{\mathsf{F}}(m_1, m_2)$$

Alternatively, a metric can be defined based on the single-link and complete-link principles [6]:

$$d_p^{\mathsf{F}}(C_1, C_2) = \frac{\min\{d_p^{\mathsf{F}}(a,b) \mid \mathcal{K} \models C_1(a) \wedge C_2(b)\}}{\max\{d_p^{\mathsf{F}}(a,b) \mid \mathcal{K} \models C_1(a) \wedge C_2(b)\}}$$

3 Optimization

Although the measures could be implemented according to the definitions, their effectiveness and also the efficiency of their computation strongly depends on the choice of the feature committee (*feature selection*). Indeed, various optimizations of the measures can be foreseen as concerns their parametric definition.

Among the possible committees, those that are able to better discriminate the objects in the ABox ought to be preferred:

Definition 3.1 (good feature set). *Let* $\mathsf{F} = \{F_1, F_2, \ldots, F_m\}$ *be a set of concept descriptions. We call* F *a* good feature set *for the knowledge base* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ *iff* $\forall a, b \in \mathsf{const}(\mathcal{D})\ \exists i \in \{1, \ldots, m\} :\ \pi_i(a) \neq \pi_i(b)$.

Note that, when the function defined in the previous section adopts a good feature set, it has the properties of a metric on the related instance-space.

Since the function strongly depends on the choice of concepts included in the committee of features F, two immediate heuristics can be derived:

1. controlling the number of concepts of the committee (which has an impact also on efficiency), including especially those that are endowed with a real discriminating power;
2. finding optimal sets of discriminating features of a given cardinality, by allowing also their composition employing the specific refinement operators.

Both these heuristics can be enforced by means of suitable ILP techniques especially when knowledge bases with large sets of instances are available. Namely, part of the entire data can be drawn in order to induce optimal F sets, in advance with respect to the application of the measure for other specific purposes as those mentioned above. The adoption of genetic programming has been considered for constructing optimal sets of features. Yet these algorithms are known to suffer from being possibly caught in local minima. An alternative may consist in employing a different probabilistic search procedure which aims at a global optimization. Thus a method based on simulated annealing [7] has been devised, whose algorithm is reported in Fig. 1.

Essentially the algorithm searches the space of all possible feature committees starting from an initial guess (determined by MAKEINITIALFS($\mathcal{K}$)) based on the concepts (both primitive and defined) currently referenced in the knowledge base. The loop controlling the search is repeated for a number of times that depends on the temperature which gradually decays to 0, when the current committee can be returned. The current feature set is iteratively refined calling a suitable procedure RANDOMSUCCESSOR(). Then the fitness of the new feature set is compared to that of the current one determining the increment of energy ΔE. If this is positive then the candidate committee replaces the current one. Otherwise it will be replaced with a probability that depends on ΔE.

As regards the heuristic FITNESSVALUE(F), it can be computed as the average *discernibility factor* [11] of the objects w.r.t. the feature set. For example, given a set of objects $IS = \{a_1, \ldots, a_n\} \subseteq \mathsf{const}(\mathcal{D})$ the fitness function may be defined:

$$\text{FITNESSVALUE}(\mathsf{F}) = k \cdot \sum_{1 \leq i < j \leq n} \sum_{h=1}^{m} \mid \pi_h(a_i) - \pi_h(a_j) \mid$$

where k is a normalization factor which may be set to: $(1/m)\,(n \cdot (n-1)/4 - n)$, depending on the number of couples of different instances that really determine the fitness measure.

As concerns finding candidates to replace the current committee (RANDOMSUCCESSOR()), the function was implemented by recurring to simple transformations of a feature set:

- adding (resp. removing) a concept C: $\mathsf{nextFS} \leftarrow \mathsf{currentFS} \cup \{C\}$ (resp. $\mathsf{nextFS} \leftarrow \mathsf{currentFS} \setminus \{C\}$)
- randomly choosing one of the current concepts from $\mathsf{currentFS}$, say C, and replacing it with one of its refinements $C' \in \text{REF}(C)$

Refining concept descriptions is language-dependent. For the adopted clausal logic, various refinement operators have been proposed in the literature [9]. *Complete* operators are to be preferred to ensure exploring the whole search-space.

FeatureSet OPTIMIZEFEATURESET($\mathcal{K}, \Delta T$)
input $\mathcal{K}$: Knowledge base
ΔT: function controlling the decrease of temperature
output FeatureSet
local currentFS: current Feature Set
nextFS: next Feature Set
Temperature: controlling the probability of downward steps
begin
currentFS $\leftarrow$ MAKEINITIALFS($\mathcal{K}$)
for $t \leftarrow 1$ **to** ∞ **do**
Temperature $\leftarrow$ Temperature $- \Delta T(t)$
if (Temperature = 0)
return currentFS
nextFS $\leftarrow$ RANDOMSUCCESSOR(currentFS,$\mathcal{K}$)
$\Delta E \leftarrow$ FITNESSVALUE(nextFS) $-$ FITNESSVALUE(currentFS)
if ($\Delta E > 0$)
currentFS $\leftarrow$ nextFS
else *// replace FS with given probability*
REPLACE(currentFS, nextFS, $e^{\Delta E / \mathsf{Temperature}}$)
end

Fig. 1. Feature Set optimization based on a Simulated Annealing procedure

4 Experiments on Similarity Search

In order to prove the effectiveness of the distance coupled with the optimization procedure, an experimentation was performed on the task of *similarity search* [16], i.e. searching instances that can be answers to relational queries by means of a notion of distance. We intended to evaluate both the effectiveness of the distance and the impact of its optimization phase.

To this purpose, a relational kNN algorithm was devised, similar to RIBL [2], with a voting procedure weighted by the distance of the query instance from its neighbors. The Java implementation exploits external Prolog libraries[2] for the reasoning services required for determining the distance between individuals.

Four relational datasets from very different domains were selected: a small one was artificially generated for the PHASE TRANSITION [3], (problem `pt4444`), the University of Washington CSE dept. dataset (UW-CSE) [13], one from the *Mutagenesis* datasets [15], and one concerned the layout structure of scientific papers (SCI-DOCS) [4]. The details[3] about these datasets are reported in Tab. 1. A simple discretization had to be preliminary operated on the numerical attributes, if present. for the measure currently does not handle these cases. Hence, the number of concepts was increased w.r.t. the original dataset.

[2] JPL 3. See `http://www.swi-prolog.org`

[3] As stated in Sect. 2, *concepts* correspond to unary predicates while predicates with larger arity are generically referred to as *relations*. *Individuals* correspond to the objects denoted by the constant names, i.e. the resources to be searched.

Table 1. Details about the datasets that were employed in the experiments

dataset	#concepts	#relations	#individuals
PHASE TRANSITION	1	4	400
UW-CSE	9	20	2208
SCI-DOCS	30	9	4585
MUTAGENESIS	68	2	9292

Table 2. Experimental results: cardinality of the induced feature set and average outcomes ($\pm$ standard deviation)

dataset	$\lvert F \rvert$	%correct	%false pos.	%false neg.
PHASE TRANSITION	6	99.97 ± 0.13	0.00 ± 0.00	0.03 ± 0.13
UW-CSE	9	99.01 ± 1.92	0.05 ± 0.08	0.94 ± 1.94
SCI-DOCS	5	85.49 ± 9.06	1.66 ± 1.87	12.85 ± 8.96
MUTAGENESIS	11	98.68 ± 1.92	0.08 ± 0.12	1.24 ± 1.94

We intended to assess the accuracy of the answers obtained inductively from the kNN procedure compared to the correct (deductive) ones. Preliminarily, an optimal distance was obtained using the procedure described in the previous section, to be employed both for selecting the nearest neighbors and for determining their weights. A 5% sample of instances was drawn from the dataset for performing the distance optimization task finding a proper feature set.

In the successive phase, a number of 20 queries (clauses whose head defines a new concept) were randomly generated provided that they had non-empty answer sets for the head variable. Then, search was simulated by testing class-membership w.r.t. the query concepts employing the kNN procedure based on the distance. The experiment was repeated applying a 10-fold cross-validation setting.

In all of the runs the number of nearest neighbors k that determine the classification of the test instance was set to $\sqrt{|\mathsf{TrSet}|}$, where TrSet is the training set related to the given fold. The cardinality of the committees determined in the first phase and the average results of the classification are reported in Table 2.

We note that the performance is quite good with a decay for the case of the SCI-DOCS dataset, which is determined by the larger variance: some queries were perfectly answered while some yielded poorer results. In terms of retrieval measures, we can say that the procedure suffers more in terms of recall rather than precision. the good results were probably due to the regularity of the information in the various datasets: for each individual the same amount of information is known, which helps to discern among them. More sparsity (incomplete information) would certainly decrease the distance acuity and, hence, the overall performance to the task.

The good performance on such datasets, despite some of them are known to be particularly difficult for learning methods, is due to the fact that for the considered task a characterization of an unknown concept is not to be learnt. Rather, it is an input of the inductive procedure based on discriminative features that help to discern between member and non-member instances.

Table 3. Experimental results (no optimization phase): average outcomes of the experiment

dataset	%correct	%false pos.	%false neg.
PHASE TRANSITION	N/A	N/A	N/A
UW-CSE	94.88	0.7	4.42
SCI-DOCS	81,29	2,65	16,06
MUTAGENESIS	94.76	0.55	4,69

It is also possible to compare the number of new features induced for the distance measure and the overall number of (primitive or defined) concepts in the KB. In the case of the MUTAGENESIS dataset, it needed only about 15% of the available concepts. However it is to be admitted that many of them had been added to the original dataset during the discretization process.

Employing smaller committees (with comparable performance results) is certainly desirable for the sake of an efficient computation of the measure. In order to asses the potential of the measure when employing basically the concepts already contained in the knowledge base, the same experiment with the same settings (10-fold cross validation) was repeated with no preliminary optimization phase; instead, for the comparison, we randomly selected the same amount of pre-defined concepts in the knowledge base as the size of the optimal set (indicated in the second column of Tab. 2). The average results obtained are reported in Tab. 3. The PHASE TRANSITION dataset had only one predefined concept all predicates are relations with arity ≥ 2) which excluded it from the possibility of a comparison.

The performance in terms of time was quite satisfactory, however various optimizations can be implemented for this specific search-task such as, for instance, computing and storing the distances in appropriate data structures [16] (e.g. *kD-trees* or *ball-trees*) that may speed-up the overall retrieval process.

5 Conclusions and Ongoing Work

In the line of past works on distance-induction, we have proposed the definition of a family of semi-distances over the instances in a clausal knowledge base. The measures are parameterized on a committee of concepts that can be selected by the proposed randomized search method.

Possible subsumption relationships between clauses in the committee may be explicitly exploited in the measure for making the relative distances more accurate. The extension to the case of concept distance may also be improved. Particularly, the measure should be extended to cope with numeric information which abounds in biological/chemical datasets.

The measures may have a wide range of application in distance-based methods to knowledge bases. Currently we are exploiting the measures in conceptual clustering algorithms where clusters will be formed by grouping instances on the grounds of their similarity assessed through the measure, triggering the induction of new emerging concepts.

Another possibility is also the extension to learning relational kernels which encode a notion of similarity, as in kFOIL [8], where measure induction and performance evaluation are intertwined.

References

[1] Bisson, G.: Learning in fol with a similarity measure. In: Swartout, W. (ed.) Proceedings of the 10th National Conference on Artificial Intelligence, pp. 82–87. MIT Press, Cambridge (1992)
[2] Emde, W., Wettschereck, D.: Relational instance-based learning. In: Saitta, L. (ed.) Proceedings of the 13th International Conference on Machine Learning, ICML 1996, pp. 122–130. Morgan Kaufmann, San Francisco (1996)
[3] Esposito, F., Ferilli, S., Fanizzi, N., Basile, T., Di Mauro, N.: An exhaustive matching procedure for the improvement of learning efficiency. In: Horváth, T., Yamamoto, A. (eds.) ILP 2003. LNCS (LNAI), vol. 2835, pp. 112–129. Springer, Heidelberg (2003)
[4] Esposito, F., Ferilli, S., Fanizzi, N., Basile, T., Di Mauro, N.: Incremental learning and concept drift in INTHELEX. Journal of Intelligent Data Analysis 8(1/2), 133–156 (2004)
[5] Hutchinson, A.: Metrics on terms and clauses. In: van Someren, M., Widmer, G. (eds.) ECML 1997. LNCS, vol. 1224, pp. 138–145. Springer, Heidelberg (1997)
[6] Jain, A., Murty, M., Flynn, P.: Data clustering: A review. ACM Computing Surveys 31(3), 264–323 (1999)
[7] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220(4598), 671–680 (1983)
[8] Landwehr, N., Passerini, A., De Raedt, L., Frasconi, P.: kFOIL: Learning simple relational kernels. In: Proceedings of the 21st National Conference on Artificial Intelligence, AAAI 2006, AAAI Press, Menlo Park (2006)
[9] Nienhuys-Cheng, S.-H., de Wolf, R.: Foundations of Inductive Logic Programming. LNCS (LNAI), vol. 1228. Springer, Heidelberg (1997)
[10] Nienhuys-Cheng, S.-H.: Distances and limits on herbrand interpretations. In: Page, D.L. (ed.) ILP 1998. LNCS, vol. 1446, pp. 250–260. Springer, Heidelberg (1998)
[11] Pawlak, Z.: Rough Sets: Theoretical Aspects of Reasoning About Data. Kluwer Academic Publishers, Dordrecht (1991)
[12] Ramon, J., Bruynooghe, M.: A framework for defining distances between first-order logic objects. In: Technical Report CW 263. Department of Computer Science, Katholieke Universiteit Leuven (1998)
[13] Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62, 107–136 (2006)
[14] Sebag, M.: Distance induction in first order logic. In: Džeroski, S., Lavrač, N. (eds.) ILP 1997. LNCS, vol. 1297, pp. 264–272. Springer, Heidelberg (1997)
[15] Srinivasan, A., Muggleton, S., King, R., Sternberg, M.: Mutagenesis: ILP experiments in a non-determinate biological domain. In: Wrobel, S. (ed.) Proceedings of the 4th International Workshop on Inductive Logic Programming, ILP 1994, number 237 in GMD-Studien. pp. 217–232 (1994)
[16] Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. Springer, Heidelberg (2007)

Clustering Relational Data Based on Randomized Propositionalization

Grant Anderson and Bernhard Pfahringer

Department of Computer Science, University of Waikato, Hamilton, New Zealand

Abstract. Clustering of relational data has so far received a lot less attention than classification of such data. In this paper we investigate a simple approach based on randomized propositionalization, which allows for applying standard clustering algorithms like KMeans to multi-relational data. We describe how random rules are generated and then turned into boolean-valued features. Clustering generally is not straightforward to evaluate, but preliminary experimental results on a number of standard ILP datasets show promising results. Clusters generated without class information usually agree well with the true class labels of cluster members, i.e. class distributions inside clusters generally differ significantly from the global class distributions. The two-tiered algorithm described shows good scalability due to the randomized nature of the first step and the availability of efficient propositional clustering algorithms for the second step.

Keywords: clustering, propositionalization, randomization.

1 Introduction

Clustering is a process by which instances are divided into groups, appropriate groupings being determined by some distance measure. Relational clustering applies this process to relational data. Such distance measures are more complex to determine for relational data than for propositional data, as relational data cannot easily be fitted to a Euclidean framework, and therefore use inter-instance similarity to determine clusters. RDBC [8], for example, uses the distance measure of RIBL [3, 5], which recursively compares the relational elements of the data until features can be propositionally compared. [6] describes a metric for terms and clauses, and relational distance measures can also be derived from relational kernels [4, 14].

Clustering of relational data has so far received a lot less attention than classification of such data. One approach based on a relational clustering tree as a variant of the relational tree learner TILDE is described in [1]. In this paper, we present a two-tiered approach to relational clustering that obviates the need for a relational distance measure, allowing us to apply standard propositional clustering algorithms to multi-relational data. In the first step we propositionalize [9] the relational data using randomly generated first-order rules (similar to the relational association rules generated by Warmr [7]), which are then converted

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 39–48, 2008.

into boolean features, based on their coverage. The generation process restricts the rules to be within certain coverage minima and maxima to avoid overly specific or general rules, respectively. The rules are also generated in a manner that encourages even coverage across the data. In the second step, the resulting propositional dataset is clustered using a standard propositional clusterer such as KMeans [10].

The next section will detail the algorithm, Section 3 reports on experiments, and the final section summarizes and outlines future work.

2 Method

The RRC (Randomized Relational Clustering) algorithm comprises two tiers: a first level which generates random rules aiming to cover all examples as uniformly as possible, and a second level which turns these rules into boolean features for a propositional representation which acts as input for any propositional clustering algorithm.

The experiments reported below employed standard KMeans using standard Euclidean distance.

Random rules are generated in the following way: at each stage a literal or test is chosen uniformly at random with the following restriction: for a literal exactly one variable/argument must be an already present variable, all others will be new variables. Tests on the other hand may not add any new variables. Tests include the usual equal and not-equal comparisons to other variables or theory constants, as well as range comparisons for numeric arguments.

To ensure that the randomly generated rules actually allow for clustering, some constraints are imposed on the generation process: rules must cover a user-defined minimum number of examples and may not cover more than a user-defined maximum, either. This prevents against both very specific and also against very general rules; worst cases would be universally true rules or rules covering to just a single example. These constraints operate on individual rules. Furthermore examples should be covered by roughly equal numbers of rules. This "uniformity of coverage" is a ruleset-level constraint. To obtain such uniform coverage, random rules are generated in small batches, then the most uniformity-preserving non-zero subset of such a batch of rules is added to the current ruleset. The basic algorithm for RRC is given in Algorithm 1.

The complexity of RRC is the sum of the complexity of both stages. Usually, when using propositionalization in ILP, the propositionalization stage dominates the total complexity, and this is true for RRC as well. Even though generating a random rule is extremely fast, its coverage still has to be determined both for checking the coverage constraints and uniformity of coverage, as well as to generate the propositional data-set. In the worst case this coverage computation can be exponential, even for a single rule. The complexity of propositional clustering algorithms on the contrary is often linear or quadratic at worst. Still, in practice we find that RRC enjoys very acceptable runtimes, and at times,

Algorithm 1. Pseudocode for the RRC algorithm

```
while Number of rules in ruleset is less than the minimum do
  while Number of rules in batch is less than the minimum do
    Generate a Rule
    if Rule is within coverage constraints then
      Add Rule to rule batch
    end if
  end while
  Calculate the most uniform subset of rules in the current rule batch
  Add those rules to the ruleset
end while
use ruleset to generate boolean-valued propositional dataset
apply any propositional clustering algorithm
```

especially for larger rulesets, the propositional clustering can actually dominate over the propositionalization stage.

3 Experiments

An evaluation of RRC on several datasets has been conducted, always generating random rules for the full dataset, and then clustering the resulting propositional data using KMeans [10] using Euclidean distance. The following standard ILP datasets were used: Mutagenesis (with and without regression-unfriendly instances), Musk1, Cancer (using only the Atom and Bond tables) and Diterpenes. For Mutagenesis and Cancer we only use low-level structural information as represented by atoms and bonds, we do not include any global properties (e.g. lumo or logP) nor predefined functional groups. In the case of the Diterpenes, three versions were generated: all pairwise combinations of the three largest classes called 3, 52, and 54.

RRC is compared to two other relational clustering approaches. RSD [16] like RRC can generate boolean-valued propositional datasets which can then be clustered by standard KMeans. Contrary to RRC's random heuristic approach, RSD generates rules via systematic search. RSD usually produces a smaller number of rules than RRC, which can be attributed to RSD's non-duplication and connectivity requirements. The minimum coverage for RSD's features was set to 25% of the number of instances in the dataset, just like for RRC. In a second experiment coverage rates for RSD were determined such that RSD would roughly generate the same number of unique rules as RRC does. The second system to compare RRC to is the Relational KMeans (RKM) algorithm of RelWeka [15], which implements the RIBL [5] distance measure, a proper distance for relational data. Whereas RSD and RRC are very similar, RKM is a rather different approach on more direct relational clustering, which does not rely on propositionalization.

To study the influence of the number of clusters that number was varied from 2 up to 50. There exists no single universally agreed upon measure for clustering quality. As true class labels are available for all datasets, which are not used during

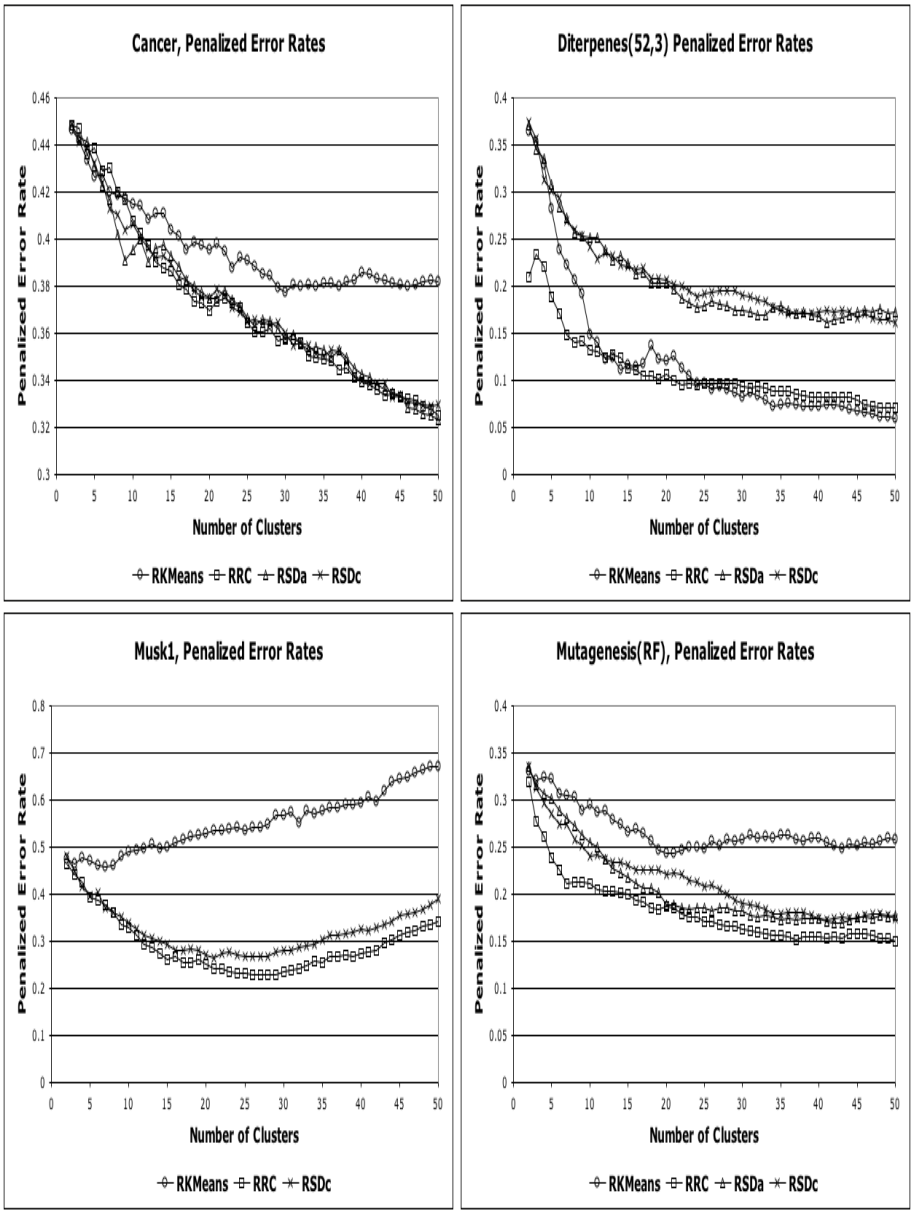


Fig. 1. Penalized Error Rates

clustering, one possible measure of cluster quality is the agreement of clusters with classes. Clearly one would expect better accuracies with more clusters, as it should be easier to find smaller class-pure clusters than larger ones. One caveat here is that

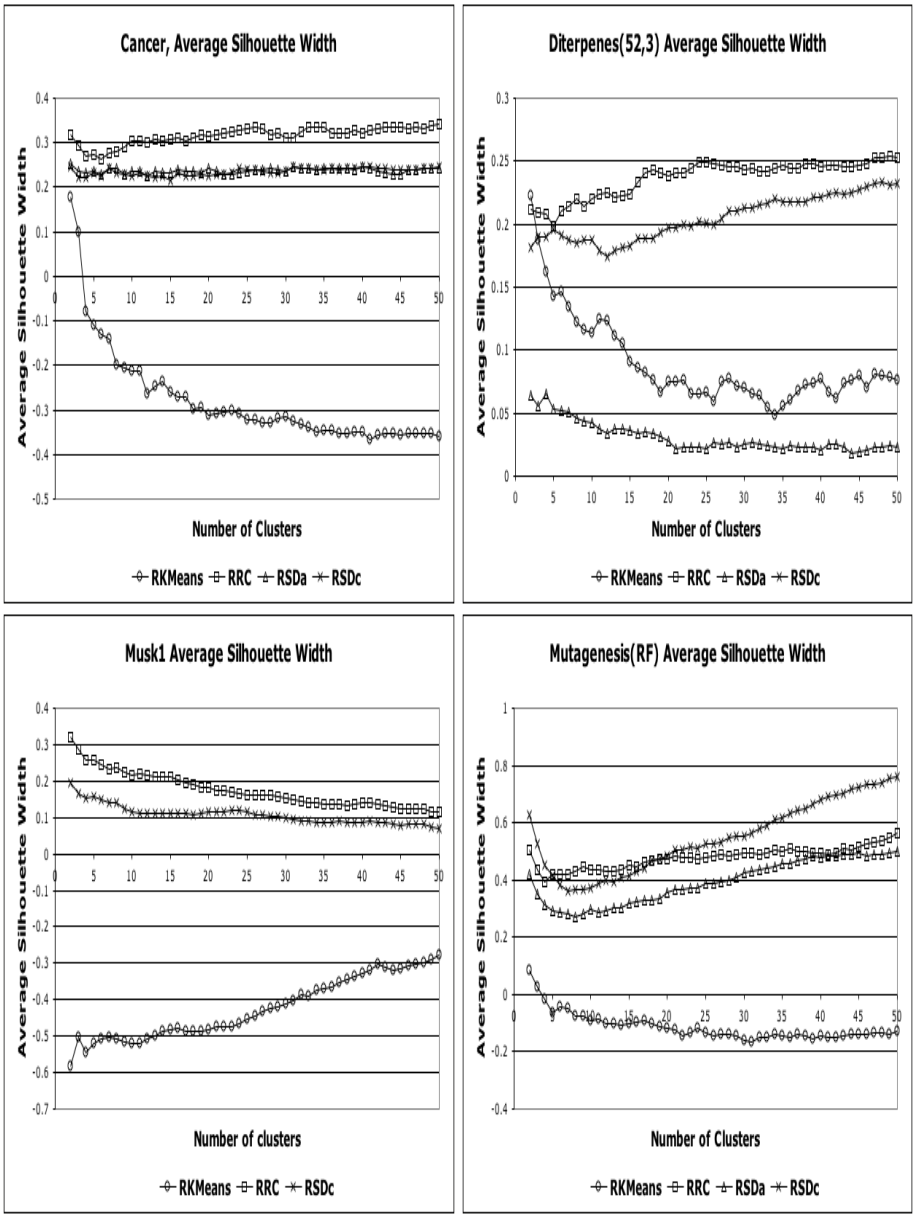


Fig. 2. Average Silhoutte Widths

when taking the majority class of each cluster as its "label", clusters with only one example will automatically be correct. Therefore a Penalized Error Rate was used that treats instances in single-instance clusters as errors.

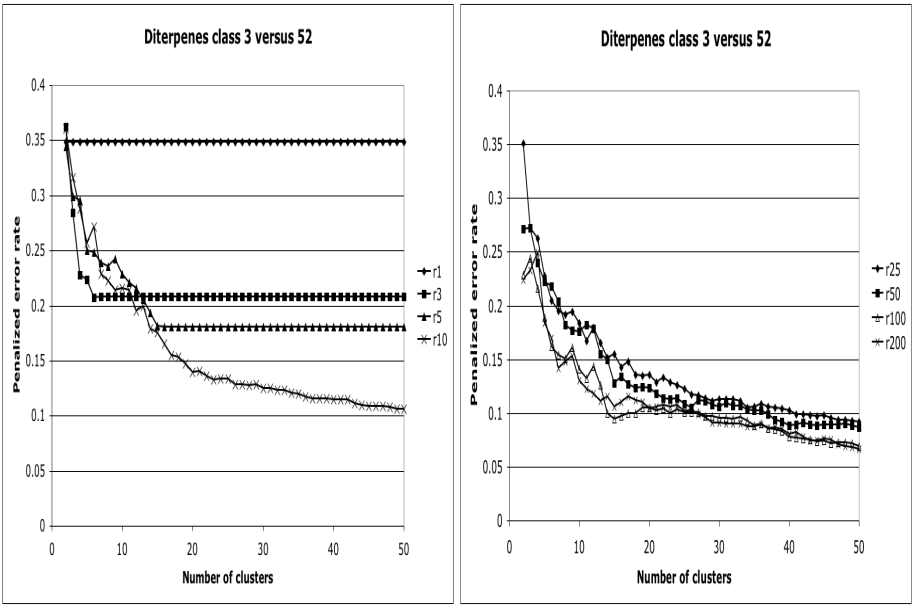


Fig. 3. Penalized error rates for various RRC rule set sizes

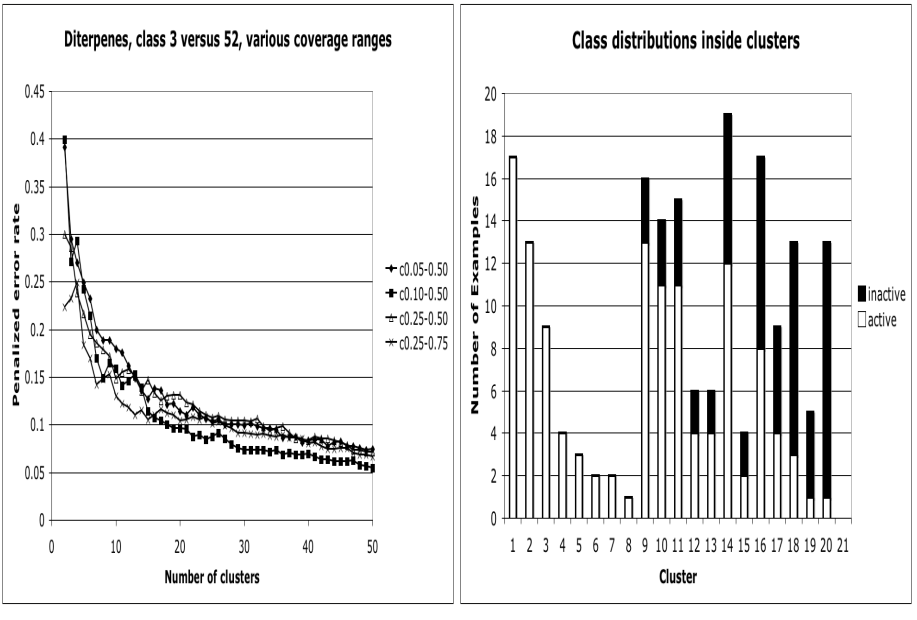


Fig. 4. Error rates for different coverages, and the class distribution across 20 clusters for MutaRF

The other measure used to compare clusterings is the average silhouette width [13], which is the average of the silhouette values for all instances in a dataset. The silhouette value s_i for an instance i is calculated according to the following formula:

$$s_i = \frac{b_i \text{ - } a_i}{\max(a_i,b_i)} \tag{1}$$

Where:
$a_i = md(i,c_i)$ (c_i = the cluster containing i)
$b_i = \min(md(i,c_{j\neq i}))$ for all clusters j not containing instance i
$md(i,c)$ = the mean distance from instance i to all instances in cluster c

Silhouette values lie between -1 and +1, with lower values indicating an increasing likelihood that the instance could have been better placed in the cluster represented by b. In the case where a cluster contains only one instance, the silhouette value of that instance is defined to be zero, again in an attempt to avoid overly positive evaluation of single-instance clusters.

For measuring cluster qualities all clusterers were run ten times with different random seeds, and the averaged results are reported. The results of this evaluation for four of the seven datasets are depicted in Figures 1 and 2. The other datasets show very similar qualitative behaviour. RSDa denotes the results for RSD datasets with a number of attributes matching the number of unique attributes RRC generates, while RSDc denotes the results for RSD datasets with a minimum coverage matching that of RRC.

The trends visible for penalized error rates follow our expectations – higher number of clusters lead to smaller penalized error rates, except on the Musk1 dataset, where its small number of instances leads to higher numbers of single-instance clusters as the number of clusters increases. Indeed, the penalized error rate begins to increase at around 30 clusters for Musk1. It should be noted that the penalized error rates for RRC are actually quite competitive to error rates that have been reported in the literature for relational classifiers on these datasets, except for Musk1.

For Cancer, Mutagenesis(RF), Mutagenesis(All) and Musk1, RRC and RSD produce fairly similar error rates, but RRC performs better on the three Diterpenes subsets. A possible reason for RSD's lower performance on Diterpenes could be the specific way background knowledge was formulated for RSD which seems to not produce some well-performing rules as found by RRC. Conversely, RelKMeans and RRC produce fairly similar results on the Diterpenes subsets, but RelKMeans performs worse on the remaining datasets. This is at least partly due to RelKMeans' tendency to produce more single-instance clusters than either of the other two algorithms.

The average silhouette width for RRC and RSD (with 25% minimum coverage) tends to increase as the number of clusters increases. On Musk1 the opposite occurs, as increasing numbers of single-instance clusters actually cause the

silhouette width to decrease. RelKMeans generally produces much lower silhouette values than the other two algorithms, which may also be related to the number of single-instance clusters produced: not only do the instances in a single-instance cluster have a silhouette value of zero themselves, they can also significantly lower the silhouette width of instances in larger clusters that lie in close proximity. Additionally the silhouette values for RelKMeans might also be affected by specific properties of its non-Euclidean distance measure. A closer investigation of this problem is left to future work.

To study the sensitivity of RRC to its user-settable parameters, a series of experiments were performed varying both the number of rules used as well as the minimum and maximum coverage values for single rules. Here we present only results for one problem: class 3 versus class 52 of the Diterpenes. Qualitatively all other sets show similar behavior, but with different optimal parameter values. Figure 3 shows the behavior of RRC for larger and larger number of rules. Numbers up to 10 are clearly not sufficient, as the error-levels quickly flatten out, not improving for larger number of clusters. Larger sizes show better performance. For this specific dataset 100 rules seem to perform slightly better than 200 rules, indicating that overfitting could be an issue for RRC.

The left figure in Figure 4 shows the effects of varying coverage limits. Again for this dataset having too low a minimum, which allows for very specialized rules to be created, seems to hurt performance. Rules that have to cover at least 10% and not more than half of all the data perform best. But the default setting of 25% to 75% does well, too.

To get a further insight into the quality of clustering, the right figure in Figure 4 depicts the class distribution for a particular 20 cluster partition of the 188 regression-friendly compounds from the mutagenicity dataset using only 10 random rules. Still, 8 of the 20 clusters are class-pure, all for the active class, though. Two of the random features generated are:

```
active(MolId) :-
        bond(MolId,_,_,2),
        atom(MolId,_,_,27,_).

active(MolId) :-
        atom(MolId,AtomId1,_,QuantaType1,Charge),
        Charge >= 0.172,
        Atom(MolId,AtomId2,_,QuantaType2,_),
        AtomId2 != AtomId1,
        QuantaType1 == QuantaType2.
```

Respectively, they represent compounds with at least one double bond plus an atom of Quanta type 27, as well as compounds with two distinct atoms of the same Quanta type, where one must have a charge of at least 0.172. Picking e.g. cluster number 4, which comprises four examples of the same class, their boolean feature values are:

```
1  2  3  4  5  6  7  8  9 10
f, t, f, f, t, t, t, t, t, t  example1
f, t, f, f, t, t, t, t, t, t  example2
f, t, f, f, t, t, f, t, t, t  example3
f, t, f, f, t, t, t, t, t, t  example4
```

Notice that these four examples are almost identical under this propositionalization, with only one exception for attribute 7 for `example3`. Figure 5 shows the structure formulas for these four compounds, and indeed three of the four are almost identical, only one nitro-group is positioned differently for each of them, and the fourth compound (`example3`, third from the left) is also very similar in structure to the other three.

4 Summary and Future Work

Relational clustering has not received much attention in ILP so far. This paper has described a two-tiered approach based on randomized propositionalization and an arbitrary propositional clustering algorithm and compared the results to two other approaches to relational clustering. The experimental results reported above look promising. This research will be extended into various directions. First and foremost comparisons to more standard clustering approaches are needed. Kernels for relational data [4] could be used together with clustering algorithms like KernelKMeans [2]. The rule generation process could be replaced by either a relational association rule finder like WarmR [7], or class-blind variants of relational rule learners like Foil [12] or Progol [11]. Lastly, this approach to class-blind propositionalization might also be useful for classification problems, especially in lazy or semi-supervised settings, as the generation process guarantees good coverage of all data including the unlabeled portion of it. More work is also needed to determine the suitability of this approach for different types of data. All experiments reported here used data-sets comprising distinct examples with no linkage between single examples. There are applications where links between examples can carry essential information. RRC will have to be evaluated for such data as well.

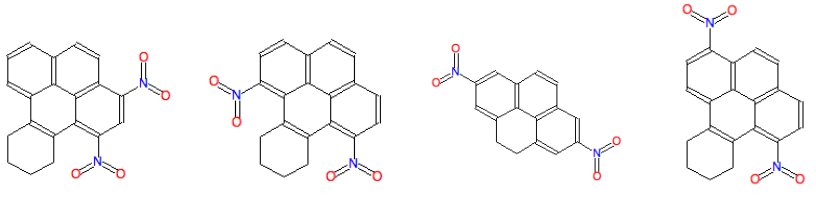


Fig. 5. The four, all active compounds of cluster 4

References

[1] Blockeel, H., De Raedt, L., Ramon, J.: Top-down induction of clustering trees. In: Proceedings of the 15th International Conference on Machine Learning, pp. 55–63 (1998)
[2] Camastra, F., Verri, A.: A Novel Kernel Method for Clustering. IEEE Transactions on Pattern Analysis and Machine Intelligence 27(5), 801–804 (2005)
[3] Emde, W., Wettschereck, D.: Relational instance-based learning. In: Proceedings of the 13th International Conference on Machine Learning, pp. 122–130 (1996)
[4] Gärtner, T., Lloyd, J.W., Flach, P.A.: Kernels and Distances for Structured Data. Machine Learning 57 (2004)
[5] Horváth, T., Wrobel, S., Bohnebeck, U.: Relational Instance-Based Learning with Lists and Terms. Machine Learning 43, 53–80 (2001)
[6] Hutchinson, A.: Metrics on terms and clauses. In: Proceedings of the 9th European Conference on Machine Learning, pp. 138–145 (1997)
[7] King, R.D., Srinivasan, A., Warmr, L.D.: A Data Mining Tool for Chemical Data Journal of Computer Aided Molecular Design. 15, 173–181 (2001)
[8] Kirsten, M., Wrobel, S.: Relational Distance-Based Clustering. In: Proceedings of the 8th International Workshop on Inductive Logic Programming, pp. 261–270 (1998)
[9] Kramer, S., Lavrac, N., Flach, P.: Propositionalization Approaches to Relational Data Mining. Relational Data Mining. Springer, Heidelberg (2001)
[10] MacQueen, J.B.: Some Methods for classification and Analysis of Multivariate Observations. In: Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, pp. 281–297 (1967)
[11] Muggleton, S.: Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming 13(3-4), 245–286 (1995)
[12] Quinlan, J.R.: Learning logical definitions from relations. Machine Learning 5, 239–266 (1990)
[13] Rousseeuw, P.J.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis Journal of Computational and Applied Mathematics, vol. 20, pp. 53–65 (1987)
[14] Woźnica, A., Kalousis, A., Hilario, M.: Kernels over Relational Algebra Structures. In: Ho, T.-B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 588–598. Springer, Heidelberg (2005)
[15] Woźnica, A., Kalousis, A., Hilario, M.: Distance and (Indefinite) Kernels for Sets of Objects. In: Perner, P. (ed.) ICDM 2006. LNCS (LNAI), vol. 4065, Springer, Heidelberg (2006)
[16] Zelezny, F., Lavrac, N.: Propositionalization-Based Relational Subgroup Discovery with RSD. Machine Learning 62(1-2), 33–63 (2006)

Structural Statistical Software Testing with Active Learning in a Graph

Nicolas Baskiotis and Michele Sebag

CNRS – INRIA – Université Paris-Sud
LRI Bat 490, F-91405 Orsay, France
Nicolas.Baskiotis, Michele.Sebag@lri.fr

Abstract. Structural Statistical Software Testing (SSST) exploits the control flow graph of the program being tested to construct test cases. Specifically, SSST exploits the *feasible paths* in the control flow graph, that is, paths which are actually exerted for some values of the program input; the limitation is that feasible paths are massively outnumbered by infeasible ones. Addressing this limitation, this paper presents an active learning algorithm aimed at sampling the feasible paths in the control flow graph. The difficulty comes from both the few feasible paths initially available and the nature of the feasible path concept, reflecting the long-range dependencies among the nodes of the control flow graph. The proposed approach is based on a frugal representation inspired from Parikh maps, and on the identification of the conjunctive subconcepts in the feasible path concept within a Disjunctive Version Space framework. Experimental validation on real-world and artificial problems demonstrates significant improvements compared to the state of the art.

Keywords: Structured Sampling, Structured Active Learning, Structural Statistical Software Testing, Disjunctive Version Space, Machine Learning Application to Computer Science.

1 Introduction

Autonomic Computing is becoming a new application domain for Machine Learning (ML), motivated by the increasing complexity of current systems [1]. Ideally, systems should be able to automatically adapt, maintain and repair themselves; a first step to this end is to build self-aware systems, using ML to automatically model the system behavior. Similar trends are observed in the field of software design; various ML approaches have been proposed for Software Testing [2,3], Software Modeling [4] and Software Debugging [5].

Resuming an earlier work [3], this paper is motivated by Statistical Structural Software Testing (SSST) [6]. SSST exploits the control flow graph of the program being tested (Fig. 1) to construct test cases; specifically, test cases are derived from the *feasible paths* in the control flow graph, that is, paths which are actually exerted for some values of the program input. However, for reasonable size programs there is a huge gap between the syntactical description of the program (the control flow graph) and its semantics (the feasible paths). In practice,

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 49–62, 2008.

the fraction of feasible paths might be as low as 10^{-5} for small size programs, making it inefficient to uniformly sample the paths in the control flow graph.

The characterization of the feasible path region faces several difficulties. First of all, the target concept (i.e. the feasible path region) is non-Markovian: a path is infeasible as it violates some subtle, long-range dependencies among the program nodes. A frugal propositional representation extending Parikh maps [7] was proposed in [3], allowing one to express node dependencies in a compact way. However, using either a relational or a propositional representation, supervised learning was found to fail; this failure was blamed on the very few feasible paths initially available, due to their high computational cost.

Meanwhile, SSST is primarily interested in acquiring more feasible paths, suggesting that an active learning approach [8] might be more relevant than a supervised learning one. In [3], a probabilistic generate-and-test approach called *EXIST* (*Exploration–Exploitation Inference for Software Testing*), built on the top of the extended Parikh map representation was proposed to generate new feasible paths. The limitation of this approach is due to the highly disjunctive nature of the target concept, blurring the probability estimates. In the current paper, the latter limitation is addressed using a bottom-up algorithm inspired from the Version Space [9], called *MLST* for *ML-based Sampling for Statistical Structural Software Testing*, which identifies the conjunctive subconcepts in the target concept. Empirical validation on real-world and artificial problems shows that *MLST* significantly improves on the state of the art.

The paper is organized as follows. Section 2 introduces the formal background and prior knowledge related to the SSST problem; it discusses the limitations of supervised learning for SSST and describes the extended Parikh representation. Section 3 gives an overview of the *MLST* algorithm. Section 4 reports on the empirical validation of *MLST* on real-world and artificial problems, and discusses the approach compared to the state of the art. The paper concludes with some perspectives for further research.

2 Position of the Problem

This section introduces statistical structural software testing (SSST), situates the problem in terms of supervised learning, and describes the extended Parikh map representation used in the following.

2.1 Statistical Structural Software Testing

Many Software Testing methods are based on the generation of test cases, where a test case associates a value to every input variable of the program being tested. For each test case, the program output is compared to the expected output (e.g., determined after the program specifications) to find out misbehaviors or bugs in the program implementation. The test quality thus reflects the coverage of the test cases (see below). Statistical testing methods, enabling intensive test campaigns, most often proceed by sampling the input space. However, uniform

sampling is bound to miss the exception branches (e.g. calling the division routine with denominator = 0), the measure of which is null. More generally, uniformly sampling the input domain does not result in a good coverage of the execution paths of the program. In order to overcome this limitation, a method combining statistical testing and structural analysis, based on the control flow graph of the program being tested (Fig. 1) was proposed by [6].

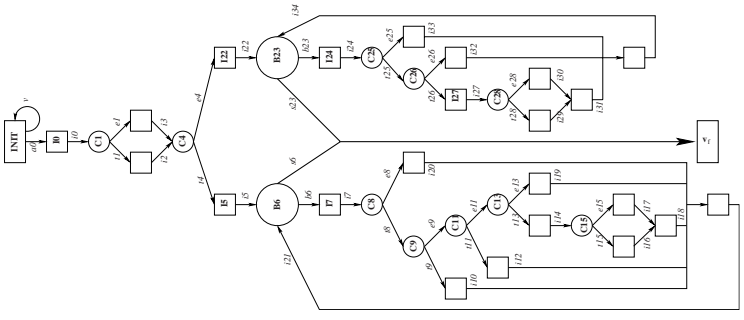


Fig. 1. Program FCT4 includes 36 nodes and 46 edges

The control flow graph provides a syntactical representation of the program. Formally, the control flow graph is a Finite State Automaton (FSA) noted $(\Sigma, \mathcal{V})$ where:

- Σ is the set of program nodes, a node being either a condition or a block of instructions, and
- $\mathcal{V}$ specifies the allowed transitions between the nodes.

For every node v in Σ, $Suc(v)$ denotes the set of successors of v, i.e. the set of all nodes w such that transition (v, w) belongs to $\mathcal{V}$. A program path noted s is represented as a finite length string on Σ, obtained by iteratively choosing a node among the successors of the current node until reaching the final node v_f.

The semantics of the program is expressed by the fact that not every path in the FSA is *feasible*, i.e. is such that the path is actually executed for some values of the program input variables. The infeasibility of a given path arises as it violates some dependencies between different parts of the program or it does not comply with the program specifications. Two most general causes for path infeasibility are the **XOR** and the **Loop** patterns.

XOR pattern. Given a program where two `if` nodes are based on some (unchanged) expression, the successors of these nodes will be correlated in every feasible path: if the successor of the first `if` node is the `then` (respectively, `else`) node, then the successor of the second `if` node must be the `then` (resp. `else`) node. Such patterns, referred to as *XOR* patterns, express the possibly long-range dependencies between the fragments of the program paths.

Loop pattern. The number of times a loop is executed happens to be restricted by the semantics of the application; e.g. when the problem involves 18 or 19

uranium beams to be controlled, the control procedure will be executed exactly 18 or 19 times [10]. This pattern is referred to as *Loop* pattern.

An upper bound T on the length of the considered paths is set by the software testing expert for practical reasons, although path length is usually unbounded (since programs generally involve `repeat` and `while` instructions). Thanks to this upper bound, one can use well-known results from labelled combinatorial structures [11] to uniformly sample the T-length paths in the control flow graph [6]. Eventually, every path is rewritten as a Constraint Satisfaction Problem, expressing the set of conditions on the input variables of the program ensuring that the path is exerted. If the Constraint Solver (CS) finds a solution, the path is labelled *feasible* and the solution precisely is the test case; otherwise the path is *infeasible*.

As already mentioned, the main limitation of this approach is when the fraction of feasible paths is tiny, which is the general case for medium length programs [6]. In such cases, the number of retrieved test cases remains insufficient while the computational effort dramatically increases; it needs some days of computation to find out a few dozen or hundred test cases. The software testing expert then inspects the program, manually decomposing the control flow graph and/or adding conditions in order to make it easier to find feasible paths.

2.2 SSST and Supervised Learning

In order to support Statistical Structural Software Testing, one possibility is to use supervised Machine Learning, exploiting a sample of labelled paths as training set. From such a training set $\mathcal{E} = \{(s_i, y_i), s_i \in \Sigma^T, y_i \in \{-1, +1\}, i = 1 \ldots, n\}$, where s_i is a path with length at most T and y_i is 1 iff s_i is feasible, supervised ML can be made to approximate the program semantics, specifically to construct a classifier predicting whether some further path is *feasible* or *infeasible*. Such a classifier would be used as a pre-processor filtering out the paths that are deemed infeasible and thus significantly reducing the CS computational cost.

In a supervised learning perspective, the SSST application presents some specificities. Firstly, it does not involve noise, i.e. the oracle (constraint solver) does not make errors[1]. Secondly, the complexity of the example space is huge with respect to the number of available examples. In most real-world problems, Σ includes a few dozen symbols; a few dozen paths are available, each a few hundred symbols long. The number of available paths is limited by the labelling cost, i.e. the runtime of the constraint solver (on average a few seconds per program path). Thirdly, the data distribution is severely imbalanced (infeasible paths outnumber the feasible ones by several orders of magnitude). Lastly, the label of a path depends on its global structure; many more examples would be required to identify the desired long-range dependencies between the transitions, within a Markovian framework. Specifically, probabilistic FSAs and likewise simple Markov models can hardly model the infeasibility patterns such as the XOR

[1] Three classes should be considered (feasible, infeasible and undecidable) as in all generality the CSPs are undecidable. However the undecidable class depends on the constraint solver and its support is negligible in practice.

or Loop patterns. While Variable Order Markov Models [12] could accommodate such patterns, they are ill-suited to the sparsity of the initial data available.

In summary, supervised learning is impaired by the poor quality of the available datasets relatively to the complexity of the instance space.

2.3 Extended Parikh Representation

A frugal and flexible representation inspired by Parikh maps was proposed in [3] in order to characterize conjunctions of XOR and Loop patterns in a compact way. Parikh maps [7,13] characterize a string from its histogram with respect to alphabet Σ; to each symbol v in Σ is associated an integer attribute $|\cdot|_v$ defined on Σ^*, where $|s|_v$ is the number of occurrences of v in string s.

As this representation is clearly insufficient to account for long range dependencies in the strings, additional attributes are defined. To each pair (v, i) in $\Sigma \times \mathbb{N}$ is associated an attribute $|\cdot|_{v,i}$, from Σ^* onto Σ, where $|s|_{v,i}$ is the successor of the i-th occurrence of the v symbol in s, or v_f if the number of v occurrences in s is less than i. Extended Parikh maps have a low representation

Table 1. Extended Parikh representation. An example

$$\begin{array}{rll} v \in \Sigma & |\cdot|_v : \Sigma^* \mapsto \mathbb{N} & |s|_v = \#v \ in \ s \\ (v,i) \in \Sigma \times \mathbb{N} & |\cdot|_{v,i} : \Sigma^* \mapsto \Sigma & |s|_{v,i} = \ \text{successor of i-th occurrence of } v \end{array}$$

$s = vwvtxytx \quad \rightarrow$

	$\lvert\cdot\rvert_v$	$\lvert\cdot\rvert_w$	$\lvert\cdot\rvert_t$	...	$\lvert\cdot\rvert_{v,1}$	$\lvert\cdot\rvert_{v,2}$	$\lvert\cdot\rvert_{w,1}$	$\lvert\cdot\rvert_{t,1}$
s	2	1	2		w	t	v	x

complexity; the number of propositional attributes is $|\Sigma| \times k$ where $k << T$ is the maximal number of occurrences of any symbol in a T-length string.

However, supervised learning within extended Parikh maps fails too. A partial conclusion is that supervised ML requires more feasible paths than normally available in SSST problems. Meanwhile, SSST is also primarily interested in building feasible paths. A new learning goal is thus defined.

3 Overview

This section describes the *MLST* system aimed at the generation of new feasible paths based on the initial training set $\mathcal{E}$. "Feasible path" and "positive example" are interchangeably used in the remainder of the paper.

3.1 Principle

Every new path s is constructed iteratively; s is initialized to the start symbol (the root node of the control flow graph); at each time step a new symbol is selected and concatenated to s. Let v be the current last symbol in s, and

denote $s.w$ the concatenation of s and w. In each time step one should select the symbol w such that $s.w$ is the prefix of many feasible paths; formally:

$$\begin{aligned} Select\ w^* &= argmax\{p_w,\ w \in Suc(v)\} \\ p_w &= Pr(s'\ feasible\ |Prefix(s') = s.w) \end{aligned} \tag{1}$$

However, the above criterion suffers from two limitations. Firstly, the set of strings s' with prefix $s.w$ is almost always empty after the first iterations (due to the size of the training set $\mathcal{E}$); and in the first iterations, this criterion would lead to duplicate the known feasible paths whereas the goal is to find *new* feasible paths. This first limitation was addressed as i) the conditioning on $Prefix(s') = s.w$ was replaced by a generalization thereof, and ii) an ϵ-greedy selection was used (section 3.3).

The second limitation comes from the fact that, after prior knowledge (section 2.1) the feasible path concept involves the conjunction of quite a few XOR patterns. With respect to the extended Parikh map representation, the target concept tc thus involves the disjunction of many conjunctive subconcepts:

$$tc = C_1 \vee \ldots \vee C_K$$

When several s' belonging to different C_i are used to estimate p_w, this estimate can be misleading; mixing the evidence derived from paths belonging to different C_i does not provide reliable indications, for the same reason as selecting the attribute with maximal entropy in a decision tree usually is inappropriate when learning a disjunctive concept. In the *EXIST* algorithm [3], this limitation was partly addressed by using the *Seeding* heuristics, stochastically extracting subsets of positive examples such that their least general generalization does not cover any negative example, referred to as admissible subsets; in each step, conditional probabilities p_w are computed from a single admissible subset. The rationale for the *Seeding* heuristics is that an admissible subset should mostly contain positive examples belonging to the same subconcept C_i; in practice, this heuristics was found to significantly improve the *EXIST* performances [3]. However, the *Seeding* heuristics suffers from two limitations. On the one hand, the initial negative examples are insufficient and do not prevent the admissible subsets from spanning over several subconcepts C_i, thus corrupting the p_w; on the other hand, the *Seeding* heuristics tends to oversample the subconcepts C_i which are best represented in the training set.

The *MLST* algorithm addresses both above limitations through a principled characterization of all subconcepts C_i represented in the training set, through the Init module (section 3.2).

Finally, *MLST* is organized as follows. The Init module (section 3.2) aims at a maximally specific disjunctive description of the initial feasible paths (the S set, in terms of Disjunctive Version Space); it constructs conjunctive subconcepts $\hat{C}_1, \ldots \hat{C}_J$, where with high probability each $\hat{C}_i$ is a specialization of some C_j represented in the training set[2]. The Generalization module (section 3.4)

[2] The identification of conjunctive subconcepts not represented in the training set is left for further study.

independently generalizes each $\hat{C}_i$. Both modules rely on the Constrained Exploration Module (section 3.3). Both Init and Generalization modules interact with the Oracle (the constraint solver), labelling every newly generated path as *feasible* or *infeasible*.

3.2 Init Module

The Init module is inspired from the Version Space framework [9]. Let the binary predicate $\mathcal{R}(s, s')$ be defined as true iff both s and s' belong to some conjunctive subconcept C_i. The Init module thus computes a stochastic estimate of $\mathcal{R}(s, s')$ noted $\hat{\mathcal{R}}(s, s')$, and uses it to construct cliques of the positive examples. With high probability (depending on the accuracy of $\hat{\mathcal{R}}$, see below), all examples in such a clique belong to the same C_i; therefore their least general generalization (lgg) defines a specialization of C_i, noted $\hat{C}_i$.

The construction of relation $\hat{\mathcal{R}}$ proceeds as follows. By definition, $\mathcal{R}(s, s')$ holds iff $lgg(s, s')$ is correct, i.e. does not cover any infeasible path. Prior knowledge on the problem domain suggests that the target concept has a tiny and fragmented coverage (section 2.1); therefore, if $\mathcal{R}(s, s')$ does not hold, then any path generated in $lgg(s, s')$ will be infeasible with high probability. Accordingly, a stochastic approximation of $\mathcal{R}(s, s')$ is implemented (Fig. 3.2), calling the Constrained Exploration Module to independently generate and label p paths in $lgg(s, s')$. If all p paths are feasible, $\hat{\mathcal{R}}(s, s')$ returns true, otherwise it returns false and the infeasible paths are added to the set $\mathcal{E}^-$ of infeasible paths. Clearly $\hat{\mathcal{R}}(s, s')$ implements a complete but incorrect approximation of $\mathcal{R}(s, s')$; its accuracy (1 - $\Pr(\hat{\mathcal{R}}(s, s')|\neg\mathcal{R}(s, s'))$) goes to 0 exponentially with p; a typical value for p in the experiments (section 4) is $p = 2$.

After $\hat{\mathcal{R}}(s, s')$ has been computed for all pairs of training feasible paths, the maximal clique $\widehat{C}(s)$ covering each feasible training path s (not already covered) is computed using a standard greedy algorithm (Fig. 3). At the j-th step, $\mathcal{V}_j$ includes all examples related by $\hat{\mathcal{R}}$ to all elements in S_j (with $S_0 = \{s\}$). If $\mathcal{V}_j$ is

```
Input: set E⁻ of infeasible paths.
    Parameter p ∈ ℕ
For all s'' ∈ E⁻
    If s'' is covered by lgg(s, s')
        return False
For i = 1 to p
    s''= C.Exploration (lgg(s, s'))
    If (label(s'') = infeasible)
        Add s'' to E⁻
        Return False
Return True
```

Fig. 2. Routine $\hat{\mathcal{R}}(s, s')$

```
S_0 = {s}
j = 1
V_j = {s'/R̂(s', s'') for all s'' ∈ S_{j-1}}
Degree_j(s') = |{s'' in V̂_j/R(s', s'')}|
While V_j is not empty
    s' = argmax_{V_j}{Degree(s'')}
    S_j = S_{j-1} ∪ {s'}
    V_{j+1} = V_j \ {s'}
    Increment j
EndWhile
Return S_j
```

Fig. 3. Routine $\widehat{C}(s)$

empty, stop; otherwise, the example in $\mathcal{V}_j$ related to most examples in $\mathcal{V}_j$ by $\hat{\mathcal{R}}$ is selected, added to S_j and removed from $\mathcal{V}_j$. Finally, the Init module produces a set of cliques noted $\hat{C}_i$. It is straightforward to show that with high probability, for each C_k represented in the training set there will be some $\hat{C}_i$ such that $\hat{C}_i$ is a specialization of C_k; the probability exponentially increases with the number of training examples in C_k and parameter p used to compute $\hat{\mathcal{R}}$.

By abuse of notations, any clique $\hat{C}$ is viewed as both a set of feasible paths and their lgg.

3.3 Constrained Exploration Module

Given a constraint h on paths, the Constrained Exploration Module aims to generate a path s such that $h(s)$ holds. This module is called by the Init module using a syntactic constraint ($h(s) \equiv s$ belongs to $lgg(s', s'')$) and in the Generalization module using a semantic constraint ($h(s) \equiv s$ is feasible).

Along the same lines as in section 3.1, the Constrained Exploration Module proceeds iteratively, initializing path s to the starting symbol and selecting in each time step the successor of the last symbol in s noted v, in order to maximize the probability for s to ultimately satisfy h. While one might want to select w maximizing the frequency of $h(s')$ over all strings s' with prefix $s.w$, (eq. 1), in most cases there is no such s'. The conditioning on $Prefix(s') = s.w$ is thus relaxed. Let v and i respectively denote the last symbol in s, and its number of occurrences ($|s|_v = i$). Condition $Prefix(s') = s.w$ is generalized as: s' is such that the successor of the i-th occurrence of the v symbol is w and the number of occurrences of w in s' is strictly greater than in s ($|s'|_{v,i} = w)AND(|s'|_w > |s|_w)$. If such paths s' exist among the available ones (training examples and examples generated along the process), frequency q_w is defined as:

$$q_w = Pr(h(s') \mid ((|s'|_{v,i} = w)AND(|s'|_w > |s|_w))) \tag{2}$$

Note that, as only standard programs are considered, any symbol (program node) has at most two successors. Letting w and w' denote the two successors of the last node v, the node selection routine thus considers three cases:

- If both q_w and $q_{w'}$ are defined, the node with maximal frequency is selected (select $argmax\{q_w, w \in Suc(v)\}$;
- If neither q_w nor $q_{w'}$ is defined, one node is selected randomly;
- Otherwise (say that q_w is defined and $q_{w'}$ is not):
  - $*$ in the Init framework, w is selected;
  - $*$ in the Generalization framework, an ϵ-greedy selection is used: w is selected with probability $1 - \epsilon$ and w' is selected with probability ϵ.

Obviously, this procedure does not guarantee that path s will satisfy $h(s)$; however, as every newly generated path is added to the available examples, and accordingly bias the computation of q_w, a fast convergence toward paths complying with constraint h was empirically observed (see section 4).

3.4 Generalization Module

The Generalization module aims at maximally generalizing every $\hat{C}$ produced by the Init module, by generating new paths s "close" to $\hat{C}$ and adding them to $\hat{C}$ if they are labelled feasible.

This module exploits the Constrained Exploration Module with constraint $h(s) \equiv s$ is feasible, with a single difference: q_w is estimated from i) only feasible paths satisfying $\hat{C}$; ii) only infeasible paths generated when generalizing $\hat{C}$. This restriction in the computation of q_w overcomes the limitations discussed in section 3.1, due to the disjunctive nature of the target concept.

Several heuristics have been investigated as alternatives to the ϵ-greedy selection in the Constrained Exploration Module; for instance, a more sophisticated Exploration vs Exploitation trade-off based on the multi-armed bandit UCB algorithm [14] was considered; however, UCB-like approaches were penalized as the "reward" probability is very low (being reminded that the fraction of feasible paths commonly is below 10^{-5}).

4 Experimental Validation

This section presents the experimental setting and goals, and reports on the results of *MLST*.

4.1 Experimental Setting

MLST is first validated on the real-world Fct4 program, including 36 nodes and 46 edges (Fig. 1). The ratio of feasible paths is circa 10^{-5} for a maximum path length $T = 250$. Fct4 is a fragment of a program used for a safety check in a nuclear plant [10].

For the sake of extensive validation, a stochastic problem generator was also designed, made of two modules. The first module defines the "program syntax", made of a control flow graph generated from a probabilistic BNF grammar[3]. The second module constructs the "program semantics", or target concept tc, determining whether a path in the above graph is feasible. After prior knowledge (section 2.1), the target concept is a conjunction of XOR patterns and Loop patterns. In order to generate satisfiable target concepts, a set $\mathcal{P}$ of paths uniformly generated from the control flow graph is first constructed; iteratively, i) one selects a XOR concept covering a strict subset of $\mathcal{P}$; ii) paths not covered by the XOR concept are removed from $\mathcal{P}$. Finally, the target concept tc is made of the conjunction of the selected XOR concepts and the Loop concepts satisfied by the paths in $\mathcal{P}$. The coverage of each conjunction is measured on an independent set of $100,000$ paths uniformly generated in the extension of tc, using [11].

[3] Three non-terminal nodes were considered (the generic structure *block*, the *if* and the *while* structures), together with two terminal nodes (the *Instruction* and the *Condition* node. The probabilities on the production rules control the length and depth of the control flow graph. Instructions are pruned in such a way that each node has exactly two successor nodes; lastly, each node is associated a distinct label.

Ten artificial problems are considered, with coverage ratio ranging in $[10^{-15}, 10^{-3}]$, number of nodes in $[20, 40]$ and path length in $[120, 250]$. Ten runs are launched for each problem, considering independent training sets, made of the set $\mathcal{E}_i$ of the initial 50 feasible paths, plus 50 infeasible paths For each $\hat{C}$ identified by the Init module, the Generalization module is launched 400 times. Set $\mathcal{E}_f$ gathers all feasible paths, the initial ones and the newly generated ones.

For each conjunctive subconcept C of the target concept represented in the training set, the performance of the algorithm is assessed by comparing the initial and final coverage of C, defined as follows. Let C_i (respectively C_f) denote the lgg of all paths in $\mathcal{E}_i \bigcap C$ (resp. $\mathcal{E}_f \bigcap C$). The initial coverage of C, noted $i(C)$, is the fraction of paths in C that belong to C_i; symmetrically the final coverage of C noted $f(C)$ is the fraction of paths in C that belong to C_f. Both $i(C)$ and $f(C)$ are estimated from a uniform sample of 10,000 examples in C, generated after [11].

For a better visualization, the average final coverage is computed using a Gaussian convolution over all subconcepts represented in the training set ($\mathcal{E}_i \bigcap C \neq \emptyset$):

$$f(x) = \frac{\sum_{C \cap \mathcal{E} \neq \emptyset} f(C) exp(-\kappa(x - i(C))^2)}{\sum_{C \cap \mathcal{E} \neq \emptyset} exp(-\kappa(x - i(C))^2)}$$

The standard deviation is similarly computed. In both cases, κ is set to 100.

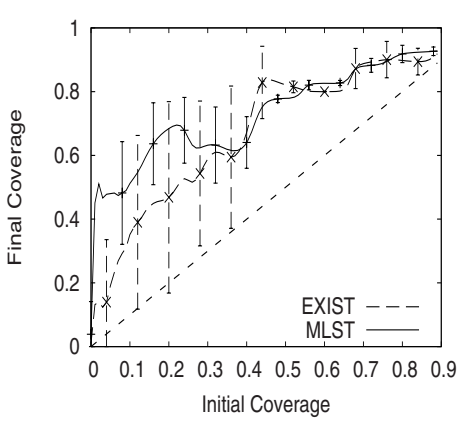
The goal of the experiments is to compare *MLST* with the former *EXIST* algorithm presented in [3] and to assess the added value of the Init module. More precisely, the performance will be examined with respect to the initial coverage of the target subconcepts in the training set.

4.2 Experimental Results

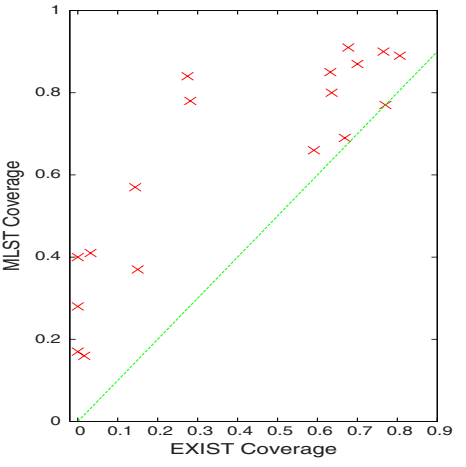
Fig. 4.(a) displays the final vs initial coverage provided by *MLST* on 10 artificial problems, using the ϵ-Greedy generalization module with $\epsilon = .5$, together with the standard deviation; complementary experiments show the good stability of the results for ϵ ranging in $[.1, .8]$. More precise results are presented in Table 2, showing that *MLST* efficiently samples the conjunctive subconcepts that are represented in the training set; when the initial coverage of the subconcept is tiny to small, the gain ranges from 5 to 2 *orders of magnitude*. A factor gain of 3 is observed when the initial coverage is between 10% to 30%.

Fig. 4.(b) reports the gain obtained on the real-world Fct4 problem comparatively to *EXIST* [3] for 10 independent runs with 3000 calls to the constraint solver. The gain of *MLST* is considered excellent by the software testing experts. The computational effort ranges from 3 to 5 minutes (on PC Pentium 3Ghz) for the Init Module and is less than 3 minutes for 400 calls to the generalization module (excluding labelling cost).

As shown in Tables 2 and 3, *MLST* significantly and consistently improves on *EXIST*. In practice, *EXIST* does not much improve the coverage of subconcepts which are poorly represented in the training set; actually, on the FCT4 problem



(a) Final vs initial coverage $\pm\sigma$ on artificial problems



(b) Comparison with *EXIST* on Fct4

Fig. 4. *MLST* (ϵ=.5): (a) Final vs initial coverage, averaged on 10 artificial problems $\times$ 10 runs; (b) Comparison with *EXIST* on the Fct4 problem (10 runs)

Table 2. *MLST*: Coverage gain averaged on 10 artificial problems $\times$ 10 runs

		$[0, 10^{-4}]$	$[10^{-4}, 10^{-3}]$	$[10^{-3}, 10^{-2}]$	$[10^{-2}, 10^{-1}]$	[.1, .3]	[.3, .6]	[.6, 1]
EXIST	$log(f/i)$	2.6 ± 2.3	2.6 ± 2.4	$1.1 \pm 1.$	$1.1 \pm 1.$			
	f/i					2.6 ± 1.8	$1.6 \pm .6$	$1.1 \pm .1$
MLST	$log(f/i)$	5.7 ± 1.2	5.3 ± 1.2	$3.7 \pm .86$	$2 \pm .72$			
	f/i					$3 \pm .1$	$1.6 \pm .3$	$1.1 \pm .1$

Table 3. Final vs initial coverage of *EXIST* and *MLST* on Fct4 averaged on 10 runs

Initial coverage	[0, .05]	[.05, .15]	[.2, .4]	[.4, .55]
EXIST final coverage	$.01 \pm .01$	$.1 \pm .06$	$.44 \pm .16$	$.71 \pm .05$
MLST final coverage	$.25 \pm .1$	$.45 \pm .07$	$.78 \pm .07$	$.83 \pm .07$

the coverage of small subconcepts ($i < .15$) is left unchanged by *EXIST* whereas *MLST* reaches a coverage of .25 for i in [0, .05] and .45 for i in [.05, .15]. This difference is explained as the stochastic *Seeding* heuristics used in *EXIST* must wait for useful negative examples to be generated, in order to construct useful admissible subsets; and in any case, it tends to increase the coverage of subconcepts that are already well represented in the training set. In contrast, *MLST* starts by characterizing all subconcepts C_i which are represented in the training set; thereafter, it spends an equal amount of time on each subconcept, resulting in consistent coverage improvements for all subconcepts, whatever their initial coverage is.

5 Discussion

Few applications of Machine Learning techniques to Software Testing have been proposed in the literature. ML has been used to feed Software Testing with program invariants [15] or characterizing relevant paths in a model checking framework [16]; it has also been used to post-process and generalize the software testing results [2].

A more remotely related work presented by [5] actually focuses on Software Debugging. Indeed, quite a few authors have investigated the generation of test cases for Software Debugging [17,18,19], most often using Constraint Satisfaction techniques.

The difference between Software Debugging and Software Testing can be characterized in terms of goal as well as quality criterion. For instance when analyzing malware, i.e. malicious software [17], the goal is to detect and prevent fatal errors. In order to do so, one must be able to run every instruction and visit every branch in the program (e.g. finding the test cases triggering malicious instructions). In other words, Software Debugging is interested in test cases enforcing a complete coverage of the block instructions, necessary and sufficient to warrant that the program is not prone to fatal errors.

In Software Testing, the goal is to certify that the software will behave according to its specifications (not every misbehavior causes a fatal error). Therefore, Software Testing aims at a complete coverage of the paths.

Indeed, when the program being tested does not involve loops, there is no difference between the path coverage and the block coverage criteria; in such cases, constraint based approaches are more efficient than ours. Otherwise, obviously the number of paths is infinite (or exponentially larger than the number of block instructions if bounded length paths are considered), and complete path coverage is not tractable. A relaxation of the complete path coverage, the goal of Software Testing thus is to uniformly sample the feasible paths.

Clearly, the distribution of the feasible paths generated by *MLST* is far from being uniform. Further work is concerned with characterizing the distribution of the generated paths, and studying its convergence.

6 Conclusion and Perspectives

The presented application of Machine Learning to Software Testing relies on an efficient representation of paths in a graph, coping with long-range dependencies and data sparsity. Further research aims at a formal characterization of the potentialities and limitations of this extended Parikh representation (see also [20]), in software testing and in other structured domains.

The main contribution of the presented work is to enable the efficient sampling of paths in a graph, targeted at some specific path region. The extension to structured domains of Active Learning, a hot topic in the Machine Learning field for over a decade [21], specifically targeted at the construction of structured examples satisfying a given property, indeed opens new theoretical and applicative perspectives to Relational Machine Learning.

With respect to Statistical Software Testing, the presented approach dramatically improves on former approaches, based on the *EXIST* algorithm [3] and on uniform sampling [6]. Further research is concerned with sampling conjunctive subconcepts which are *not* represented in the initial training set. In the longer run, the extension of this approach to related applications such as equivalence testers or reachability testers for huge automata [22] will be studied.

Acknowledgements

We thank the anonymous reviewers for many helpful comments. The authors gratefully acknowledge the support of `Pascal` Network of Excellence IS T-2002-506 778.

References

1. Rish, I., Das, R., Tesauro, G., Kephart, J.: ECML-PKDD Workshop Autonomic Computing: A new Challenge for Machine Learning (2006)
2. Bréhélin, L., Gascuel, O., Caraux, G.: Hidden Markov models with patterns to learn boolean vector sequences and application to the built-in self-test for integrated circuits. IEEE Transactions Pattern Analysis and Machine Intelligence 23(9), 997–1008 (2001)
3. Baskiotis, N., Sebag, M., Gaudel, M.C., Gouraud, S.D.: Software testing: A machine learning approach. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 2274–2279 (2007)
4. Xiao, G., Southey, F., Holte, R.C., Wilkinson, D.F.: Software testing by active learning for commercial games. In: Proceedings of the 20th National Conference on Artificial Intelligence (AAAI), pp. 898–903 (2005)
5. Zheng, A.X., Jordan, M.I., Liblit, B., Naik, M., Aiken, A.: Statistical debugging: simultaneous identification of multiple bugs. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 1105–1112 (2006)
6. Denise, A., Gaudel, M.C., Gouraud, S.D.: A generic method for statistical testing. In: Proceedings of the 15th International Symposium on Software Reliability Engineering, pp. 25–34 (2004)
7. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (1979)
8. Dasgupta, S.: Coarse sample complexity bounds for active learning. Advances in Neural Information Processing Systems, 235–242 (2005)
9. Mitchell, T.: Generalization as search. Artificial Intelligence 18, 203–226 (1982)
10. Gouraud, S.D.: Statistical Software Testing based on Structural Combinatorics. In: PhD thesis, LRI, UniversitéParis-Sud (2004)
11. Flajolet, P., Zimmermann, P., Cutsem, B.V.: A calculus for the random generation of labelled combinatorial structures. Theoretical Computer Science 132(2), 1–35 (1994)
12. Begleiter, R., El-Yaniv, R., Yona, G.: On prediction using variable order Markov models. Journal of Artificial Intelligence Research 22, 385–421 (2004)
13. Fischer, E., Magniez, F., de Rougemont, M.: Approximate satisfiability and equivalence. In: 21th IEEE Symposium on Logic in Computer Science, pp. 421–430 (2006)

14. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning 47(2-3), 235–256 (2002)
15. Ernst, M.D., Cockrell, J., Griswold, W.G., Notkin, D.: Dynamically discovering likely program invariants to support program evolution. IEEE Transactions on Software Engineering 27(2), 99–123 (2001)
16. Vardhan, A., Sen, K., Viswanathan, M., Agha, G.: Actively learning to verify safety for FIFO automata. In: Foundations of Software Technology and Theoretical Computer Science, pp. 494–505 (2004)
17. Moser, A., Krügel, C., Kirda, E.: Exploring multiple execution paths for malware analysis. In: IEEE Symposium on Security and Privacy, pp. 231–245 (2007)
18. Cadar, C., Ganesh, V., Pawlowski, P.M., Dill, D.L., Engler, D.R.: EXE: automatically generating inputs of death. In: ACM Conference on Computer and Communications Security, pp. 322–335 (2006)
19. Godefroid, P., Klarlund, N., Sen, K.: DART: directed automated random testing. In: ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 213–223 (2005)
20. Clark, A., Florencio, C.C., Watkins, C.: Languages as hyperplanes: Grammatical inference with string kernels. In: Proceedings of the 17th European Conference on Machine Learning, pp. 90–101 (2006)
21. Cohn, D.A., Ghahramani, Z., Jordan, M.I.: Active learning with statistical models. In: Advances in Neural Information Processing Systems, pp. 705–712. MIT Press, Cambridge (1995)
22. Yannakakis, M.: Testing, optimization, and games. In: Proceedings of the 31st International Colloquium on Automata, Languages and Programming, pp. 28–45 (2004)

Learning Declarative Bias

Will Bridewell[1] and Ljupčo Todorovski[1,2]

[1] Computational Learning Laboratory,
Center for the Study of Language and Information,
Stanford University, Stanford, CA, USA 94305
willb@csli.stanford.edu
[2] University of Ljubljana, Faculty of Administration
Gosarjeva 5, SI-1000 Ljubljana, Slovenia
ljupco.todorovski@fu.uni-lj.si

Abstract. In this paper, we introduce an inductive logic programming approach to learning declarative bias. The target learning task is inductive process modeling, which we briefly review. Next we discuss our approach to bias induction while emphasizing predicates that characterize the knowledge and models associated with the HIPM system. We then evaluate how the learned bias affects the space of model structures that HIPM considers and how well it generalizes to other search problems in the same domain. Results indicate that the bias reduces the size of the search space without removing the most accurate structures. In addition, our approach reconstructs known constraints in population dynamics. We conclude the paper by discussing a generalization of the technique to learning bias for inductive logic programming and by noting directions for future work.

Keywords: inductive process modeling, meta-learning, transfer learning.

1 Introduction

Research on inductive process modeling [1] emphasizes programs that build models of dynamic systems. As the name suggests, the models are sets of processes that relate groups of entities. For example, neighboring wolf and rabbit populations interact through a predation process, which may take one of many forms. As input, these programs take observations, which record system behavior over time, background knowledge, which consists of scientifically meaningful generic processes, and entities whose behavior should be explained. The output is a model that comprises processes instantiated with the available entities. A naive solution to the task would exhaustively search the space of models defined by the instantiated processes, but this approach produces several nonsensical models and the search space grows exponentially in the number of instantiations. To make inductive process modeling manageable in nontrivial domains, one must introduce bias.

Recently, researchers developed the notion of a process hierarchy to define the space of plausible model structures [2]. This solution defines which processes

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 63–77, 2008.

must always appear in a model, which ones depend on the presence of others, and which ones mutually exclude each other. Although one can use the hierarchy to substantially reduce the size of the search space, specifying relationships that both constrain the space and have validity in the modeled domain is difficult. Importantly, the introduction of this bias replaces the task of manually building a model with that of manually defining the space of plausible model structures. Ideally we would like to automatically discover this knowledge.

Ample literature exists on bias selection [3], which emphasizes search through the space of learning parameters, and constructive induction [4], which increases the size of the search space. In contrast, we wish to learn constraints that will reshape the search space and ensure that the program considers only plausible and accurate models. In the context of inductive process modeling, the learned constraints would imply the same restrictions as those encoded in the process hierarchy. For instance, in the case of the wolves and rabbits, we would like to discover that an accurate model of the dynamics must include a predation process. As this example hints, these constraints are generalizations drawn from the space of models.

To be more specific, we used inductive logic programming to find clauses that characterize accurate and inaccurate models. The examples are individual models from the search space that are classified as accurate or inaccurate according to their fit to training data. As further input, the background knowledge comprises descriptions of model organization such as predicates that indicate whether a model includes a particular process and which entities participate in a process. Given this information, we learn theories whose clauses describe accurate and inaccurate models, which we then turn into constraints on the model structures.

Before describing our approach in detail, we first provide a high level introduction of inductive process modeling with an emphasis on those aspects used to learn declarative bias. We then describe our learning algorithm, which relies on the combination of HIPM [2] and Aleph [5]. Next we describe promising results on a predator–prey domain and highlight the transfer of learned bias across multiple data sets. Following the experiments, we explain the general applicability of this approach to other artificial intelligence tasks and identify other work of a similar nature. Finally, we discuss limitations of our current approach and highlight its effect on inductive process modeling.

2 Inductive Process Modeling

Our program learns bias by analyzing the model structures that HIPM generates during search. For this reason, we provide a rough description of the knowledge representations for generic process libraries and quantitative process models. In the context of this paper, details about the internals of the processes (e.g., conditions and equations), the structural and parametric search techniques, and the simulation routine are of less importance. We will briefly mention these aspects, which are described in greater detail elsewhere [1,2].

Table 1. Part of a library for population dynamics, which includes both generic entities and processes. A generic process's type appears in braces after its name. For simplicity, we suppress each process's equations, conditions, and numeric parameters.

generic entity predator: variables concentration{sum};	generic entity prey: variables concentration{sum};
generic process exp_growth{growth}: entity E{prey, predator};	generic process log_growth{growth}: entity E{prey, predator};
generic process exp_loss{loss}: entity E{prey, predator};	generic process holling_type_1{predation}: entity P{prey}, R{predator};
generic process holling_type_2{predation}: entity P{prey}, R{predator};	generic process holling_type_3{predation}: entity P{prey}, R{predator};

At a high level, the background knowledge used by HIPM takes the form of the process library shown in Table 1. This library defines two generic entities which have properties (i.e., variables and constants). In this case, both declarations specify a single variable, *concentration*, with an additive combining scheme.[1] The generic entities fill labeled roles in the generic processes, which are defined below each process's name. For instance, *exp_growth* requires an instantiation of the generic entity *prey* or *predator* and in its complete form contains a constant that defines the growth rate and an equation that alters the prey's concentration. Notice that this generic process also has type *growth*, which is specified in braces after the name. In HIPM, process types define groups of generic processes that help delimit the search space. For instance, one could require that all population dynamics models include one of the many predation processes, letting the program select from these based on the model's fit to data.

In contrast to libraries, models contain instantiated entities and processes. For the entities, one must define the properties by specifying the values of constants and by either associating variables with a trajectory or stating their initial values. As an example, the entity *Wolf* could instantiate *predator* and its variable could refer to weekly recordings of a localized wolf population's size. Instantiating a process involves specifying the values of local constants and filling each labeled role with an entity that has the appropriate type. To a degree, and for the purposes of this paper, one can view generic processes and entities as predicates and the instantiated versions as ground facts.

Expanding on this logical view, a model is a conjunctive clause that can predict the quantitative behavior of a dynamic system. To explain a data set, HIPM constructs model structures that satisfy constraints, which are part of the bias. The program then estimates the numeric parameters of each model using a gradient

[1] When multiple processes influence a variable, one must aggregate the effects. To this end, HIPM supports combining schemes that select the minimal or maximal effect, add the effects, or multiply them.

Table 2. The logical representation of the population dynamics library from Table 1

entity(predator).	entity(prey).
process(exp_growth, growth).	process(log_growth, growth).
process(exp_loss, loss).	process(holling_type_1, predation).
process(holling_type_2, predation).	process(holling_type_3, predation).

search algorithm [6] coupled with random restarts. HIPM relies on CVODE [7] to simulate the system of equations entailed by the model and compares the trajectories to measured time-series. To learn bias, our program employs a more general logical view of the model and couples it with a predicate that associates the structure with a particular level of accuracy.

3 Learning Bias by Inductive Logic Programming

The primary contribution of this paper is a logical representation that lets one learn declarative bias by analyzing the space of possible models. We developed such a formalism for describing both libraries and models associated with inductive process modeling and we present a strategy for applying inductive logic programming tools. In this section, we describe the formalism, the input to the learning system, the performance element that uses the learned rules, and the learning software that we employed.

Model structures contain a number of processes, each of which relates a number of entities, and each of those may be shared among multiple processes. These three properties indicate the inherent relational structure of the models and suggest that a first-order representation would best capture this characteristic. Indeed, in Section 2 we suggested that the inductive process modeling representation resembles first-order logic. That is, one could view the names of generic processes and entities as predicates and instantiations of these as ground facts. However, this mapping creates a domain-specific language that one must tailor to other modeling problems. For instance, we would like to use the same predicates for describing knowledge and models from the population dynamics domain as for those in a physiological one. Therefore, we designed domain-independent predicates that explicitly characterize the structure of processes and models.

Table 2 contains an encoding of the population dynamics library from Table 1. There are only two predicates in this representation: *entity* and *process*. For our program, the only relevant information about a generic entity is its name. Generic processes differ slightly in that they have both a name, the first argument, and a type, the second one. Currently we leave the entity roles of a generic process unspecified because the instantiated processes in each model relate entities according to the type specification in the library. A key feature of this formalization is that it limits the domain-specific information to the process library and lets us build high-level predicates that generalize to other tasks.

Table 3. The logical representation of a population dynamics model

model(m1).	r2(m1, 0.85).
process_instance(m1, p1, exp_growth).	process_instance(m1, p2, exp_loss).
parameter(m1, p1, aurelia).	parameter(m1, p2, nasutum).
process_instance(m1, p3, holling_type_1).	
parameter(m1, p3, aurelia).	entity_instance(nasutum, predator).
parameter(m1, p3, nasutum).	entity_instance(aurelia, prey).

The logical representation for process models resembles that for the library with the key differences that it ties the components of individual models together and that it includes information about the process parameters. To illustrate, consider the model in Table 3. Since the examples are a collection of ground facts, we need a way to associate processes, parameters, and entities with particular models. Here, we introduce the predicate *model* that declares a ground term to be a model identifier. This term appears in all predicates associated with that particular model except for *entity_instance*. Since all models explain the behavior of the same entities, a direct tie between them is unnecessary.

Similarly, we associate parameters with particular processes. For instance, the model in Table 3 contains a *process_instance* that belongs to model *m1*, has the unique identifier *p1*, and instantiates the generic process *exp_growth*. The use of the unique identifier lets one relate multiple entities to a single process.

The collection of ground facts from Table 3 defines an organized model structure that has a particular accuracy. Here we encode this value with the *r2* predicate, which records the coefficient of determination calculated over the training data. This value falls in the range $[0, 1]$ and indicates how well the shape of the simulated trajectory matches that of the observed values.

Although the predicates for the generic process library and for instantiated models define the structures, they would lead to clauses that are difficult to interpret. To address this problem, we introduce a set of higher level predicates that *describe* the structures in terms of properties. These predicates combine those from Tables 2 and 3 into forms like those in Table 4. To illustrate, the predicate `includes_processtype_entity(m1, p1, aurelia)` compactly represents the conjunction `process_instance(m1, p1, exp_growth), process(exp_growth, growth), parameter(m1, p1, aurelia), entity_instance(aurelia, prey)`. We use these higher-level predicates *in addition* to the lower-level ones so that one can identify comprehensible rules without losing the ability to learn unanticipated relationships among model structures.

Whereas the library definition and high level predicates compose the background knowledge, the models serve as the examples. To assign the models to a target class, we use the predicates *accurate_model* and *inaccurate_model*. We

Table 4. Examples of high-level predicates that describe the model structure

includes_processtype(M, T) :– model(M), process(P, T), process_instance(M, _, P).	includes_processtype_entity(M, T, E) :– model(M), entity(E), process_instance(M, PI, P), process(P, T), parameter(M, PI, EI), entity_instance(EI, E).

define the rules for these predicates as `accurate_model(M) :- r2(M, R), R` $\geq$ `threshold.` and `inaccurate_model(M) :- r2(M, R), R` $<$ `threshold.`[2] We describe the method for selecting a threshold in Section 4.

Given background knowledge and examples, our approach produces theories whose clauses characterize the structure of accurate and inaccurate models. We can use each such clause to bias the search space of candidate model structures. To this end, the clauses that predict accurate models specify which structures should remain in the search space, while the clauses for inaccurate models state ones the automated modeler should prune. To learn these rules, we apply the inductive logic programming system Aleph, which produces separate theories for accurate and inaccurate models. For example, Aleph could learn the clause, `accurate_model(M) :- includes_processtype(M, predation).`, which we would turn into background knowledge for HIPM that forces all models to contain a process with type *predation*.

As we have described the task, learning declarative bias seems like a natural problem for inductive logic programming; however, a propositional approach would work as well since we are working in a finite domain without recursive predicates. For instance, given a fixed, finite domain of processes, we could construct a Boolean feature vector for each model that contains all the properties captured by the background knowledge. This method would effectively involve a reimplementation of the LINUS system [8], which, as a result, reinforces the appropriateness of a relational learner for inducing declarative bias. In the next section, we evaluate our described approach in an ecological domain.

4 Empirical Evaluation

To determine the utility of the approach described in Section 3, we applied it in the domain of population dynamics. After describing the experimental method, we detail a strategy for selecting an appropriate threshold value to separate accurate models from inaccurate ones. In the rest of the section, we evaluate three conjectures about the learned bias. First, we expect the bias will substantially reduce the search space of candidate model structures. Second, we expect the reduced search space will retain the most accurate models. And third, we anticipate that the learned model constraints will be consistent with the existing

[2] Note that *r2* is not a background knowledge predicate for learning bias.

knowledge in the domain of population dynamics modeling and have a potential to contribute new findings to it.

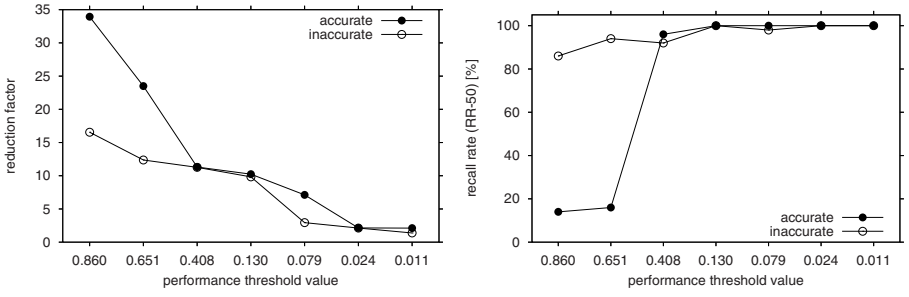
4.1 Method

We performed the experiments on three modeling tasks. Each includes modeling population dynamics from measured time-series of concentrations of two species, aurelia and nasutum [9], involved in a predator–prey interaction. We ran HIPM on the three tasks with an initial library that lack constraints on process combinations in models. HIPM performed exhaustive search through the space of 9402 model structures that have up to five processes, fit the constant parameters of each candidate structure against the measured time-series, and reported the obtained model and its performance (i.e., the coefficient of determination on the training time-series). We reformulated the traces of HIPM runs in first-order logic to create three data sets for learning bias: PP1, PP2, and PP3.

For each data set, we must first select the value of the performance threshold that distinguishes accurate models from the inaccurate ones first. We evaluate the effect of the threshold value on the recall of the best models in the next subsection, and based on this analysis, we propose a method for selecting an optimal threshold for a given data set. Having selected the threshold value for the training set, we run Aleph to induce bias, and then evaluate its performance on the remaining two data sets which were unseen during learning.[3] We use Aleph's default parameter settings, except that we set its noise parameter to 10, which lets a clause cover up to 10 negative examples, the minpos parameter to 2, which prevents Aleph from adding ground facts to the theory, and the nodes parameter to 100,000, which increases the upper bound on the program's search space complexity.

To validate our conjectures about the induced bias, we use two quantitative evaluation metrics and qualitative analysis. First, we measure the size of the search space before and after applying the learned bias. Formally, we calculate the *reduction factor* as $|SS_0|/|SS|$, where $|SS_0|$ denotes the number of models in the unconstrained search space and $|SS|$ is the number of model structures in the biased space. Second, *best model recall* measures the percentage of the n best models of a particular data set in the entire search space that also appear in the reduced space. Ideally, the reduced space would retain 100% of the best models. In the experiments, we measure recall of the ten and fifty best models. Next, we compare the distribution of the model performance in the reduced search space to the distribution of the model performance in the entire space. We expect the overall performance of models in the reduced search space to compare favorably to the overall performance of all candidate models. Finally, we present the model constraints that were learned from all three data sets and analyze them in terms of their consistency with existing knowledge in the population dynamics domain.

[3] Since HIPM performs exhaustive search, we do not run HIPM again to search the constrained space of models. Instead, we remove the model structures that violate the induced constraints.



(a) (b)

Fig. 1. Sensitivity analysis of the bias to the performance threshold value as calculated on training data. The graph on the left-hand side (a) shows the reduction of the search space, while the one on the right-hand side (b) shows the recall of the best fifty models. Full and empty circle symbols indicate the performance of the model constraints for the accurate and inaccurate class, respectively.

4.2 Selecting a Performance Threshold

Before we select an appropriate performance threshold for classifying a model as accurate or inaccurate, we analyze its effect on bias performance in terms of search space reduction and best models recall. To identify a list of plausible threshold values, we rank the models according to their accuracy and divide them into 10 bins. Then we select the point of maximal performance change between two consecutive models in each bin. This leads to an initial list of ten candidate thresholds, which we revise by removing consecutive points that are close to each other.

The graph in Figure 1(a) shows that the relation between the performance threshold and the search space reduction is strictly monotonic. As one would expect, high threshold values render most of the models inaccurate, which leads to highly specific model constraints. Using these specific model constraints considerably narrows the search space, which is reflected in the high values of the reduction factor (over 30). On the other hand, the graph in Figure 1(b) shows that the bias corresponding to high threshold values is too restrictive and removes most of the best models from the search space. Model constraints induced with threshold values of 0.860 and 0.651 filter out most of the fifty best models, while the others include most of the best models. The model constraints for the class of inaccurate models follow roughly the same pattern.

Note that the analysis performed here is limited to the training data set, to emphasize the fact that we select the performance threshold on the basis of training data only. The bias performance change on the test data sets correlates highly with the results on the training data and is virtually identical to that shown in Figure 1(b). This indicates the good generalization performance of the induced bias, which we further analyze in the next subsection.

Table 5. Evaluating the utility of the model constraints learned on a train data set TrainDS (for accurate and inaccurate target predicates) on test data sets TestDS. For each bias, the table reports the reduction of the search space (RSS) and the recall of the top ten (BMR-10) and top fifty (BMR-50) models for the test set.

Model Constraints (Bias)			Bias Evaluation		
TrainDS	Class	RSS	TestDS	BMR-10[%]	BMR-50[%]
PP1	accurate	11×	PP2	100	96
			PP3	100	94
PP1	inaccurate	11×	PP2	90	88
			PP3	90	92
PP2	accurate	16×	PP1	100	98
			PP3	60	36
PP2	inaccurate	11×	PP1	100	98
			PP3	90	82
PP3	accurate	10×	PP1	100	100
			PP2	100	100
PP3	inaccurate	9×	PP1	100	100
			PP2	100	98

In summary, the graphs in Figure 1 clearly render 0.130 and 0.408 as optimal threshold values, since they both lead to a substantially reduced search spaces that retain most of the best models. Based on this analysis, we use 0.408 as threshold for performing further experiments on the PP1 data set. The analysis of threshold influence on the bias performance on the other two data sets, PP2 and PP3, shows a similar effect.

4.3 Evaluating the Generalization Performance

Once we identify the optimal threshold value for learning bias on a particular data set, we induce the bias using that threshold, and we analyze its performance on the other two data sets. Table 5 summarizes the results of the evaluation. All model constraints, induced on different data sets and for different target predicates, lead to a reduction factor ranging from 9 to 16. The highest reduction rate is observed for the bias induced on the PP2 data set for the inaccurate class.

All induced model constraints recall most of the top ten models for all test data sets, except the ones induced on the PP2 data (inaccurate class) that recall 6 of the top ten models for PP3. The recall of the top fifty models is lower, but still over 90% for most of the cases, with the exception for the bias induced from the PP2 data set (82% and 63% recall of the top fifty models for PP3 using inaccurate and accurate constraints, respectively). The worse overall performance is observed when applying the bias induced from the accurate models in PP2 data set, which is the most restrictive bias in terms of the search space reduction.

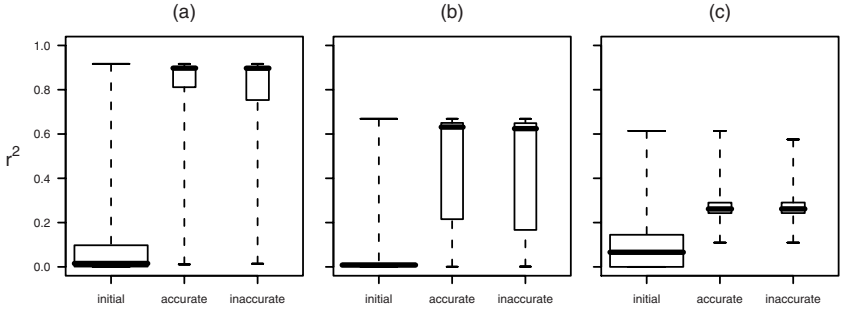


Fig. 2. Comparing the distributions of the model performance in the initial (unconstrained) search space to the performance distributions in the biased spaces induced from the PP1 data set (accurate and inaccurate class). Graphs (a), (b), and (c) compare the distributions on the PP1, PP2, and PP3 data sets respectively.

The graphs in Figure 2 compare the performance distribution for the models in the initial (unconstrained) search space to the performance distribution in the search spaces constrained by the bias induced from the PP1 data set. Figure 2(a) shows that the median r^2 in the entire search space is close to 0 denoting that most of the models perform poorly on the training time-series (recall that the range of r^2 is [0,1]). The learned bias focuses the search to the space of candidate model structures that perform much better, their median value being 0.89. Note however, that the bias does not filter out all inaccurate models from the search space, since the minimal observed performance remains close to 0. The graphs in Figures 2(b) and 2(c) show that the constraints induced on the PP1 data set also narrows the search focus on the test data sets, PP2 and PP3. Bias still shifts the distribution towards models with better performance: from median r^2 close to 0 to medians of 0.63 and 0.26, for PP2 and PP3 respectively. However, the performance distribution for PP2 is blurred towards worse performing models indicating that the bias induced on the PP1 data set allows a considerable number of sub-optimal models. Nevertheless, the comparison of the distributions confirm that the bias learned on PP1 generalizes well to PP2 and PP3. The effect of the constraints induced from PP2 and PP3 are comparable.

4.4 Semantic Analysis of the Induced Constraints

For each data set Aleph induced a different collection of model constraints. Yet, eight of these, presented in Table 6, appear in every theory. Since these constraints provide generalizations that we can match against existing domain knowledge, we analyze their potential to enrich it.

Before describing the clauses produced by Aleph, we acknowledge their propositional nature and stress its superficiality. To illustrate, we point out that the

Table 6. The eight rules that appear in each of the theories induced from the PP1, PP2, and PP3 data sets

```
accurate_model(M) :-
  includes_processtype_entity(M, loss, predator),
  includes_process_entity(M, exp_growth, prey),
  includes_process(M, hassell_varley_2).
accurate_model(M) :-
  includes_processtype_entity(M, loss, predator),
  includes_process_entity(M, exp_growth, prey),
  includes_process(M, holling_type_3).

accurate_model(M) :-
  includes_processtype_entity(M, loss, predator),
  includes_process_entity(M, log_growth, prey),
  includes_process(M, hassell_varley_2).
accurate_model(M) :-
  includes_processtype_entity(M, loss, predator),
  includes_process_entity(M, log_growth, prey),
  includes_process(M, holling_type_2).
accurate_model(M) :-
  includes_processtype_entity(M, loss, predator),
  includes_process_entity(M, log_growth, prey),
  includes_process(M, holling_type_3).

inaccurate_model(M) :-
  doesnotinclude_processtype_entity(M, growth, prey).
inaccurate_model(M) :-
  doesnotinclude_processtype_entity(M, loss, predator).
inaccurate_model(M) :-
  doesnotinclude_processtype(M, interaction).
```

higher-level predicates mask the relational structure of the rules. For example, one can rewrite the first rule in Table 6 in terms of the lower-level predicates as

```
accurate_model(M) :-
  model(M), entity(predator), process_instance(M, PI, P),
  process(P, loss), parameter(M, PI, EI),
  entity_instance(EI, predator), process(exp_growth), entity(prey),
  process_instance(M, PI2, exp_growth), parameter(M, PI2, EI2),
  entity_instance(EI2, prey), process(hassell_varley_2),
  process_instance(M, PI3, hassell_varley_2).
```

In addition, had we relaxed the limitations on which entity types could bind to particular processes (e.g., by letting entities having type *predator* bind to *growth* processes and those having type *prey* bind to *loss* processes), rules such as

```
inaccurate_model(M) :- includes_processtype_entity(M, growth, ET),
  includes_processtype_entity(M, loss, ET).
```

would likely appear.

Turning now to the semantic analysis of the rules, we see that five of the frequent clauses shown in Table 6 characterize accurate models. The first two specify that the structure of an accurate predator–prey model includes three processes: loss of the predator, exponential growth of the prey, and one interaction process, which may be one of the two specific formulations. The three other clauses that characterize accurate models are similar, since they also claim that an accurate models include three processes of loss, growth, and interaction. The difference from the first two rules is that they specify alternative form of the growth process (logistic instead of exponential) and a larger set of interactions.

Finally, the three model constraints for the inaccurate_model predicate paraphrase the rules for the class of accurate models, but they are more general. They identify the three main properties of an inaccurate model's structure: the lack of prey species growth, the lack of predator loss, and the lack of interaction between species. In other words, an accurate model structure should include at least one process of each type, which is a rediscovery of the well known fact established in early work by Lotka and Volterra [10]. On the other hand, the five rules for the accurate models establish novel hypotheses about predator–prey interaction between the observed species, which ecologists may further evaluate.

We consider the reconstruction of well-known facts from the domain of population dynamics as important evidence about our program's potential to learn useful and meaningful bias constraints. This result also improves the credibility of the hypotheses established by the other model constraints.

5 General Discussion

Although initial results suggest the feasibility of our approach to inducing bias, many questions remain. In this section, we describe related work and explore the generality and limitations of our method.

In the introduction, we differentiated our work from bias selection and constructive induction, but there are other similar approaches that we should discuss. In particular, our research falls within the general category of metalearning [11], but much of this work emphasizes the prediction of algorithm performance, whereas we use the output of learning to reshape the search space for an algorithm's future applications. Instead, our work more closely matches that of McCreath and Sharma [12] who used inductive logic programming to learn mode and type declarations, which could constrain the space of candidate clauses. Notably, their program produced syntactic constraints unrelated to the specific domain, whereas our approach induces semantic ones that are interpretable within the domain's context. Additionally, we note similarities with learning control

rules for planning [13] since the algorithms analyze the output of the planner to improve future performance. However, such systems generally view only the operators and the context of their application as opposed to an entire plan.

The predicates with which we characterize model structures resemble the relational clichés introduced by Silverstein and Pazzani [14]. Relational clichés are conjunctions of predicates that are useful for building classification rules in a particular domain. As such they relate to a combination of processes that must appear in an accurate model. While our approach learns these combinations (i.e., clichés) from examples of inaccurate and accurate models, the work presented in [14] does not deal with learning clichés but rather demonstrates the benefit of using them as declarative bias for learning classification rules. More recently, Morin and Matwin [15] proposed an approach to inducing relational clichés, but instead of using the meta-learning approach presented here, they learn clichés directly from the examples in one domain and then transfer them into another domain. Their work focuses on learning and transfer of bias between domains and not on learning constraint rules that would contribute to the theory in the domain of interest. On the other hand, transferring induced knowledge to other domains is an open challenge for our approach.

Even though we showed how to learn bias in the limited context of inductive process modeling, we expect that it will generalize to other domains. For instance, in the case of inductive logic programming, one would examine each evaluated clause as a separate entity and identify a bias that restricts the structure of the antecedents. This usage would require the learning program to report the performance of all considered clauses instead of just those in the final theory, but such an extension requires minimal effort with the potential for substantial gains in both the effectiveness of the search and the plausibility of the induced theories. Extensions to propositional and association rule learning are similar.

Apart from generalization to other artificial intelligence tasks, there are several open avenues for future work. First, in this paper, we assume that exhaustive search of the model space is possible. Such scenarios are uncommon, and we need to better understand the effects of model sampling on the induction of bias. Second, we would like to use a similar approach to analyze the best performing models in a domain. This task requires an inductive logic programming system that learns from positive examples only [16] and raises questions about what to include in the set of best models. Third, Reid [17] introduces the idea of learning an evaluation bias, which lets one infer the reliability of a logical rule from its past performance in related tasks. In the spirit of his research, we would like to estimate the quantitative fit of a model structure based upon its performance in similar domains. This step would let a program establish priorities over a set of candidate structures so that "better" ones would have earlier access to the computationally expensive parameter-estimation routine. Finally, Pazzani and Kibler [18] show that biasing the space of candidate models with domain-specific knowledge helps reduce overfitting and improves overall accuracy on a test set. We need to evaluate whether this holds true when the bias is automatically induced.

6 Conclusion

In this paper, we developed a representation that lets one learn declarative bias for inductive process modeling using tool provided for inductive logic programming. Our primary contribution is that we showed how to construct the background knowledge, how to describe the examples, and how to select a threshold for the supervised learning task. We then evaluated our approach on a population dynamics domain and found that the learned bias substantially reduced the size of the candidate model structure space. We also found that the bias increased the proportion of accurate models in both the training data and test data taken from the same domain. Importantly many of the induced constraints verified known ecological theory. Finally, we described related work, proposed the generalization of this method to other learning algorithms, and highlighted future work that will lead to a better understanding of this research area. We believe that the reported approach opens a promising new avenue for scientists in artificial intelligence that is rich with open questions.

Acknowledgments. This research was supported by Grant No. IIS-0326059 from the National Science Foundation. We thank Pat Langley, Stuart Borrett, and Tolga Könik for discussions that influenced the ideas in this paper.

References

1. Langley, P., Sánchez, J., Todorovski, L., Džeroski, S.: Inducing process models from continuous data. In: Proceedings of the Nineteenth International Conference on Machine Learning, Sydney, pp. 347–354. Morgan Kaufmann, San Francisco (2002)
2. Todorovski, L., Bridewell, W., Shiran, O., Langley, P.: Inducing hierarchical process models in dynamic domains. In: Proceedings of the Twentieth National Conference on Artificial Intelligence, Pittsburgh, PA, pp. 892–897. AAAI Press, Menlo Park (2005)
3. Provost, F., Buchanan, B.: Inductive policy: The pragmatics of bias selection. Machine Learning 20, 35–61 (1995)
4. Utgoff, P.E.: Machine Learning of Inductive Bias. Kluwer Academic Publishers, Boston, MA (1986)
5. Srinivasan, A.: The Aleph Manual. Computing Laboratory, Oxford University Press, Oxford (2000), http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph_toc.html
6. Bunch, D.S., Gay, D.M., Welsch, R.E.: Algorithm 717: Subroutines for maximum likelihood and quasi-likelihood estimation of parameters in nonlinear regression models. ACM Transactions on Mathematical Software 19, 109–130 (1993)
7. Cohen, S., Hindmarsh, A.: CVODE, a stiff/nonstiff ODE solver in C. Computers in Physics 10, 138–143 (1996)
8. Lavrac, N., Džeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood, New York (1994)
9. Jost, C., Ellner, S.: Testing for predator dependence in predator–prey dynamics: A non-parametric approach. In: Proceedings of the Royal Society of London B, vol. 267(1453), pp. 1611–1620 (2000)

10. Kingsland, S.E.: Modeling Nature, 2nd edn. The University of Chicago Press, Chicago, IL (1995)
11. Giraud-Carrier, C., Vilalta, R., Brazdil, P.: Introduction to the special issue on meta-learning. Machine Learning 54, 187–193 (2004)
12. McCreath, E., Sharma, A.: Extraction of meta-knowledge to restrict the hypothesis space for ILP systems. In: Proceedings of the Eighth Australian Joint Conference on Artificial Intelligence, Canberra, Australia, pp. 75–82. World Scientific Publishers, Singapore (1995)
13. Huang, Y., Selman, B., Kautz, H.A.: Learning declarative control rules for constraint-based planning. In: Proceedings of the Seventeenth International Conference on Machine Learning, Stanford, CA, pp. 415–422. Morgan Kaufmann, San Francisco (2000)
14. Silverstein, G., Pazzani, M.J.: Relational clichés: Constraining constructive induction during relational learning. In: Proceedings of the Eighth International Workshop on Machine Learning, pp. 203–207. Morgan Kaufmann, San Francisco (1991)
15. Morin, J., Matwin, S.: Relational learning with transfer of knowledge between domains. In: Proceedings of the Thirteenth Biennial Conference of the Canadian Society for Computational Studies of Intelligence, pp. 379–388. Springer, Heidelberg (2000)
16. Muggleton, S.: Learning from positive data. In: Proceedings of the Sixth International Workshop on Inductive Logic Programming, Stockholm, Sweden, pp. 358–376. Springer, Heidelberg (1996)
17. Reid, M.: DEFT guessing: Using inductive transfer to improve rule evaluation from limited data. Ph.D. thesis, University of New South Wales, Sydney, Australia (2007)
18. Pazzani, M.J., Kibler, D.F.: The utility of knowledge in inductive learning. Machine Learning 9, 57–94 (1992)

ILP :- Just Trie It

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

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

Abstract. Despite the considerable success of Inductive Logic Programming (ILP), deployed ILP systems still have efficiency problems when applied to complex problems. Several techniques have been proposed to address the efficiency issue. Such proposals include query transformations, query packs, lazy evaluation and parallel execution of ILP systems, to mention just a few. We propose a novel technique that avoids the procedure of *deducing* each example to evaluate each constructed clause. The technique takes advantage of the two stage procedure of Mode Directed Inverse Entailment (MDIE) systems. In the first stage of a MDIE system, where the bottom clause is constructed, we store not only the bottom clause but also valuable additional information. The information stored is sufficient to evaluate the clauses constructed in the second stage without the need for a theorem prover. We used a data structure called Trie to efficiently store all bottom clauses produced using all examples (positive and negative) as seeds. The technique was implemented and evaluated using two well known data sets from the ILP literature. The results are promising both in terms of execution time and accuracy.

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

1 Introduction

Inductive Logic Programming (ILP) [1] has been successfully applied to problems in several application domains [2]. Nevertheless, it is recognised that efficiency and scalability are major obstacles to the increased usage of ILP systems in complex applications with large hypothesis spaces. Research on improving the efficiency of ILP systems has focused on reducing their sequential execution time, either by reducing the number of hypotheses generated (see, e.g., [3,4]), or by efficiently testing candidate hypotheses (see, e.g., [5,6,7,8]). Another line of research pursued by several researchers is the parallelization of ILP systems [9].

During execution, an ILP system generates many candidate hypotheses which have a lot of similarities among them. Usually, these similarities tend to correspond to common prefixes among the hypotheses. Blockeel et al. [5] defined query-packs as a technique to exploit this pattern and improve the execution

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 78–87, 2008.

time of ILP systems. Inspired by their work, we focus on how to reduce the amount of theorem proving to a minimum. We call our novel approach *Trieing MDIE*. The key idea is to use a single *trie data structure* (also known as a *prefix-tree*) to inherently and efficiently exploit the similarities among the hypotheses, hence reducing memory usage and allowing us to store useful information about clauses. But this is as close we get to query-packs, which can be considered as a form of trie designed to improve execution speed. Instead we follow a different approach based on Mode Directed Inverse Entailment (MDIE)[10].

To explain our approach, *Trieing MDIE*, let us recall that a *traditional* MDIE-based procedure is performed in two stages. In the first stage an example is chosen and the bottom clause [10] is constructed (saturation stage). In the second stage a search is performed using the bottom clause as the lower bound of the search space. During the second stage clauses are constructed and evaluated using the examples. In the *Trieing MDIE* approach we saturate all examples (positive and negative). From each bottom clause we generate valid clauses and insert them in an *unique* trie, such that the trie contains counters describing clause coverage. The search procedure of the second stage will therefore be replaced by a simple inspection of this trie to retrieve the best clause.

Trieing MDIE can be implemented in MDIE-based ILP systems such as Progol [11], Aleph [12], Indlog [8], and April [13]. It is usable in positive only data sets and is not applicable to learn recursive theories. A further improvement in speedup can be achieved by combining *Trieing MDIE* with known strategies to parallelise ILP systems [9], such as, *parallel exploration of independent hypotheses* and *data parallelism*. Notice that tries have already been proposed previously [14] as a technique to reduce the amount of memory storage. In that study, tries were used to store the clauses constructed during the second stage of the MDIE method. They have also been used in FARMER [15] to overcome efficiency issues of the Warmr system [16] for learning Association Rules.

The remainder of the paper is organised as follows. Section 2 introduces the trie data structure and describes its implementation. In Section 3 we present the algorithm to use tries in MDIE-based ILP systems. Section 4 presents some limitations of the technique when using a background knowledge containing predicates that are not pure logic programs. In Section 5 we present an empirical evaluation of the impact in execution time of the proposed data structure. Finally, in Section 6, we draw some conclusions and propose further work.

2 The Trie Data Structure

Tries were first proposed by Fredkin [17], the original name inspired by the central letters of the word *retrieval*. Tries were originally invented to index dictionaries, and have since been generalised to index recursive data structures such as terms. Please refer to [18,19,20] for the use of tries in automated theorem proving, term rewriting and tabled logic programs. An essential property of the trie data structure is that common prefixes are represented only once. This naturally applies to ILP, as the hypothesis space is structured as a lattice and

hypotheses close to one another in the lattice have common prefixes (literals). Hence, it clearly matches the common prefix property of tries. We thus argue that, for ILP systems, this is an interesting property that we should be able to take advantage of for storing hypotheses and associated information.

Using Tries to Represent Hypotheses

A trie is a tree structure where each different path through the trie data units, the *trie nodes*, corresponds to a term. At the entry point we have the root node. Internal nodes represent symbols in terms and leaf nodes specify the end of terms. Each root-to-leaf path represents a term described by the symbols labelling the nodes traversed. Two terms with common prefixes will branch off from each other at the first distinguishing symbol. In order to maximise the number of common trie nodes when storing hypotheses in a trie, we used Prolog lists to represent the clauses corresponding to hypotheses. Figure 1 presents an example for a trie with three clauses.

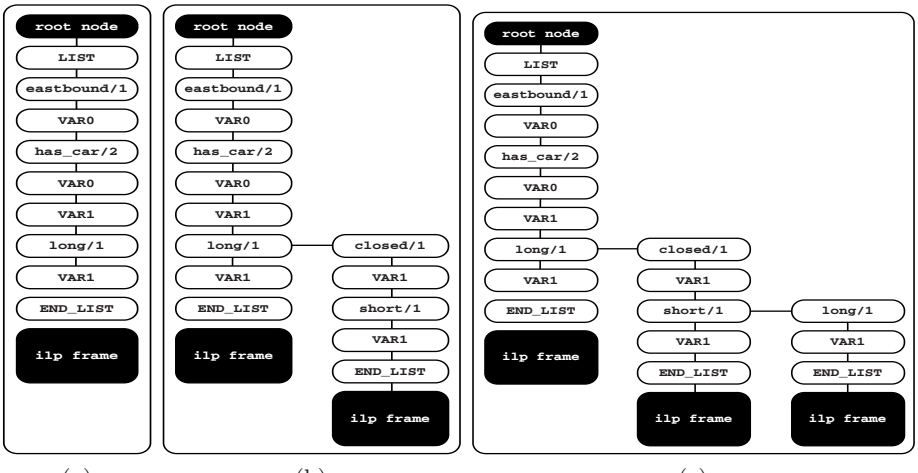


Fig. 1. Using tries to represent:
(a) *C = eastbound(T) :- has_car(T,C), long(C).*
(b) *C and D = eastbound(T) :- has_car(T,C), closed(C), short(C).*
(c) *C, D and E = eastbound(T) :- has_car(T,C), closed(C), long(C).*

Initially, the trie contains the root node only. Next, we insert the clause $[eastbound(T), has\_car(T, C), long(C)]$ and nine nodes are added to represent it (Figure 1(a)). The clause $[eastbound(T), has\_car(T, C), closed(C), short(C)]$ is then inserted which requires eleven nodes. As it shares a common prefix with the previous clause, we save the six initial nodes common to both representations (Figure 1(b)). The clause $[eastbound(T), has\_car(T, C), closed(C), long(C)]$ is

next inserted and we save more eight nodes, the same six nodes as before plus two more nodes common with the second inserted clause (Figure 1(c)).

Each path in the trie terminates at a leaf data structure, the *ilp frame* data structure, that we used to extend the original trie structure to store information associated with the clause, namely info concerning the number of positive and negative examples covered by the clause (the use of this information is discussed in more detail next). Another important point when using tries to represent clauses is the treatment of variables. We follow the formalism proposed by Bachmair *et al.* [18], where each variable in a term is represented as a distinct constant. Formally, this corresponds to a function, $numbervar()$, from the set of variables in a term t to the sequence of constants $VAR_0, ..., VAR_N$, such that $numbervar(X) < numbervar(Y)$ if X is encountered before Y in the left-to-right traversal of t. For example, in the term $[eastbound(T), has\_car(T, C), long(C)]$, $numbervar(T)$ and $numbervar(C)$ are respectively VAR_0 and VAR_1.

3 Trieing MDIE

MDIE-based systems use bottom-clauses to generate sets of clauses. Given a bottom-clause $\bot_e$, the refinement operator generates clauses from $\bot_e$ that will cover at least the example e. Let us call this set $\mathcal{S}$. The clauses in $\mathcal{S}$ share e, so we can say that e *forms* $\mathcal{S}$. Note that, in general, $\mathcal{S}$ will be arbitrarily large, and we will need to impose some further restrictions, such as clause length restrictions. Moreover, note that even if complete, $\mathcal{S}$ does not correspond to all clauses that cover e. Indeed, it is well known that the bottom-clause is not complete: we can generate clauses that cover an example e which cannot be refined from $\bot_e$ [21].

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

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

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

and the following clause c:

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

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

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

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

covered by the clause. This suggests the following approach: given an example e construct the corresponding bottom clause $\bot_e$ and generate a set $\mathcal{S}$ with all legal clauses c such that c θ-subsumes $\bot_e$. Next, given a set of examples $\{e_1^+, e_2^+, \ldots, e_n^+, e_1^-, e_2^-, \ldots, e_m^-\}$ construct the corresponding sets of clauses $\{\mathcal{S}_1^+, \mathcal{S}_2^+, \ldots, \mathcal{S}_n^+, \mathcal{S}_1^-, \mathcal{S}_2^-, \ldots, \mathcal{S}_m^-\}$. Finding the best clauses should be just a question of searching for clauses that appear in most $\mathcal{S}_i^+$ and not in $\mathcal{S}_i^-$. More precisely, if we allow no noise, then we would like to find the clause with the largest coverage from $\cup_i \mathcal{S}_i^+ \setminus \cup_j \mathcal{S}_j^-$. Note that we are not interested in the examples, but in the set of all clauses of interest, which would to a first approximation be close to $\cup_i \mathcal{S}_i^+$. Now, this set may grow quickly, and therefore needs a compact and fast representation. It makes sense to represent sets of clauses by structures optimised for quick access and sharing, such as the *tries* discussed in Section 2.

Our Algorithm

To *estimate* the coverage of all clauses we will walk over all examples $e \in E$ as follows. Visit positive examples first, and assume that we do not care about clauses that only cover negative examples:

- If $e \in E^+$ and $c \notin \mathcal{S}$, add c to $\mathcal{S}$ and state that c covers one positive example.
- If $e \in E^+$ and $c \in \mathcal{S}$, state that c covers one more positive example.
- If $e \in E^-$ and $c \in \mathcal{S}$, state that c covers one more negative example.
- If $e \in E^-$ and $c \notin \mathcal{S}$, do nothing.

We therefore need to define an abstract set, that we call *decorated set* $\mathcal{S}$, with all clauses and their coverage. A *decorated set* is a set whose elements are clauses, and attached to each element are counters (one counter for each class of the learning problem). With this abstraction we can easily implement any theory construction algorithm as shown in Figure 2.

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

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

Fig. 2. The greedy cover algorithm of a MDIE system implementation

The main difference with systems like Progol or Aleph concerns the inner procedure $learnTrieingMDIE()$. We next describe how clauses are learned in the *Trieing MDIE* algorithm (see Figure 3). The *Trieing MDIE* algorithm has two basic stages. First a decorated set $\mathcal{S}$ is constructed (lines 1 to 9) and then the best

clause (according to some metric) is found by inspection of the set (line 10). The decorated set $\mathcal{S}$ is constructed as described above. First, all positive examples are processed and then a pruning procedure, $prune()$, is invoked to remove useless clauses from $\mathcal{S}$ (e.g., clauses with coverage lower than some predefined minimum number of examples). Next, all negative examples are also processed and then the set is pruned again.

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

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

Fig. 3. The learning algorithm of *Trieing MDIE*

Figure 4 shows how the set $\mathcal{S}$ is filled for each example. First we generate the bottom clause (line 2). Then, using the bottom clause, we generate all valid clauses[4] (line 4), normalise them (line 5), and insert them in the set (line 6). Normalisation orders the literals according to the Prolog "@ <" order relation. We generate all renaming of existential variables to check if a variant already exists in the trie, therefore guaranteeing a unique representation for each clause. The $insertUpdateInSet()$ procedure works as follows. If the example class is positive, the clause is inserted into $\mathcal{S}$ and the positive counter is updated. If the example class is negative, the clause is not added to $\mathcal{S}$, only the negative counter of the clause is updated. This means that the clauses generated from the negative examples that are not in $\mathcal{S}$ are discarded.

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

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

Fig. 4. From an example to a set of clauses

[4] Clauses satisfying the language and bias constraints.

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

4 Trie the Real World

We have presented our algorithm in the context of an ideal world, where the background knowledge is a pure logic program, the saturated clause is generated to its completion, and all clauses subsuming the saturated clause are enumerated. Next we discuss how our algorithm can cope with two major issues we found in practise: completion of the saturated clause and syntactic redundancy.

Completeness and Recall Factor

In almost every data set, ILP can only generate a subset of the full saturated clause. This subset is controlled by a depth factor i on the maximum length of variable chains, and also by the *recall factor*. Next, we discuss how these two factors affect our algorithm.

The i constraint is a syntactic constraint that is applied uniformly to every goal while generating the bottom-clause. By induction, it should be clear that if a variable chain respects the i constraint in a saturated clause, it will respect the same constraint on every other saturated clause.

The *recall factor* parameter indicates how many solutions to a goal can be introduced in the bottom clause. If set to $*$, it will include every answer. On the other hand, if set to a lower threshold than the actual number of different answers a goal can generate, this parameter becomes a source of incompleteness. As the answer order will be different with different examples, using low-values of this parameter is not recommended when using the proposed algorithm.

Syntactically Redundant Clauses

The *switching lemma* tells us that if a conjunction of goals $G_1, \ldots, G_n$ is satisfiable, then any permutation of these goals is also satisfiable. ILP systems often take advantage of this principle to reduce the number of clauses they actually need to generate: if one generates $a(X), b(X)$ there is no point in also generating $b(X), a(X)$. On the other hand, traditional ILP systems cannot use any ordering of goals, as they must respect an ordering that is efficient for Prolog execution. As our algorithm does not actually evaluate goals, this is unnecessary: we can choose any ordering between goals when checking for redundant goals. In this vein, we try to simplify all syntactically redundant clauses into a normalised clauses, so that all syntactically equivalent clauses will have a canonical representation in the trie.

5 Experiments and Results

To evaluate the impact of the proposed approach we adapted the April ILP system [13] so that it could be executed with support for tries and applied the system to well known data sets. The experiments were made on an AMD Athlon(tm) MP 2000+ dual-processor PC with 2 GB of memory, running the Linux RedHat (kernel 2.4.20) operating system. The data sets used were downloaded from the Machine Learning repositories of the Universities of Oxford[5] and York[6]. Table 1 characterises the data sets in terms of number of positive and negative examples as well as background knowledge size.

Table 1. Data sets

Data Set	$\mid E^+ \mid$	$\mid E^- \mid$	$\mid B \mid$
Carcinogenesis	202	174	44
Mutagenesis	136	69	21

For each data set, the system was executed for a standard MDIE implementation using a deterministic top-down breadth-first search procedure (**dtd-bf**) and for MDIE using tries (**Trieing**). We varied the maximum depth of the clauses from 2 (one literal in the body) to 4 (3 literals in the body). Table 2 compares the execution times and the number of clauses generated by both approaches.

Table 2. Execution time and number of clauses generated

Data Set	Depth	Execution Time		Clauses Generated	
		dtd-bf	Trieing	dtd-bf	Trieing
Carcinogenesis	2	4	6	8,012	17,352
	3	56	82	233,860	684,855
	4	2,205	4,049	5,827,459	26,613,734
Mutagenesis	2	2,130	3,442	8,991	18,308
	3	13,809	5,343	339,591	834,023
	4	21,600	7,115	9,261,589	20,445,957

The results confirm that *Trieing MDIE* generates considerably more clauses (ranging from two up to five fold) than the *dtd-bf MDIE* approach. In spite of considering more clauses in the search, *Trieing MDIE* outperforms *dtd-bf MDIE* in the Mutagenesis data set. However, it is around 50% slower than *dtd-bf MDIE* in the Carcinogenesis data set. Naturally, if the same number of clauses is generated for *dtd-bf MDIE*, *Trieing MDIE* will also compare favorably.

Although the impact in execution time of *Trieing MDIE* is inconclusive, the impact in accuracy is promising. In Table 3 we can observe that *Trieing MDIE* achieves very good results, both in terms of accuracy and memory usage.

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

Table 3. Accuracy and memory usage (in each cell the 3 values represent clause length of 2/3/4)

Data set	Accuracy		Memory	
	dtd-bf	Trieing	dtd-bf	Trieing
Carcinogenesis	72 / 48 / 51	72 / 62 / 69	19 / 19 / 122	19 / 21 / 59
Mutagenesis	65 / 71 / 74	65 / 94 / 82	10 / 19 / 99	13 / 13 / 22

6 Conclusions

This paper is a novel contribution to the effort of improving ILP systems efficiency. A novel technique was put forward to reduce execution time of MDIE-based ILP systems. This improvement is achieved by avoiding the theorem proving of all clauses constructed during the search stage of a MDIE system. This was possible by using a trie data structure to store all generated clauses, and their coverage. Tries take advantage of common pre-fixes in clauses which leads to a quite small memory requirements for the ILP system. Coverage information allows the system to estimate efficiently the value of clauses.

The proposed technique was integrated in an ILP system implemented in Prolog and empirically evaluated on three well known data sets. The results indicate a significant reduction in execution time (for the same number of clauses evaluated) in all data sets used. The results also indicate an increase in accuracy since the system performs wider searches. Overall the amount of memory used to analyse the data sets was very small.

In the future we plan to extend the evaluation process. We will first determine the degree of non-determinism of the background knowledge of each data set. We expect the result to improve with an increase of non-determinism of the predicates in the background knowledge (more effort in theorem proving). To show further the advantage in memory savings we intend to use much larger data sets. Data from the ILP challenge from 2005, for example, will be considered.

Acknowledgements

This work has been partially supported by Fundação para a Ciência e Tecnologia and by project Myddas (POSC/EIA/59154/2004). Nuno A. Fonseca is funded by FCT grant SFRH/BPD/26737/2005.

References

1. Muggleton, S.: Inductive logic programming. New Generation Computing 8(4), 295–317 (1991)
2. Ilp applications. http://www.cs.bris.ac.uk/~ILPnet2/Applications/

3. Nédellec, C., Rouveirol, C., Adé, H., Bergadano, F., Tausend, B.: Declarative bias in ILP. In: De Raedt, L. (ed.) Advances in Inductive Logic Programming, pp. 82–103. IOS Press, Amsterdam (1996)
4. Camacho, R.: Improving the efficiency of ilp systems using an incremental language level search. In: Annual Machine Learning Conference of Belgium and the Netherlands (2002)
5. 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. Journal of Artificial Intelligence Research 16, 135–166 (2002)
6. Costa, V.S., Srinivasan, A., Camacho, R.: A note on two simple transformations for improving the efficiency of an ILP system. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, Springer, Heidelberg (2000)
7. Costa, V.S., Srinivasan, A., Camacho, R., Hendrik, Van Laer, W.: Query transformations for improving the efficiency of ilp systems. Journal of Machine Learning Research (2002)
8. Camacho, R.: Inductive Logic Programming to Induce Controllers. In: PhD thesis, Univerity of Porto (2000)
9. Fonseca, N.A., Silva, F., Camacho, R.: Strategies to Parallelize ILP Systems. In: Kramer, S., Pfahringer, B. (eds.) ILP 2005. LNCS (LNAI), vol. 3625, pp. 136–153. Springer, Heidelberg (2005)
10. Muggleton, S.: Inverse Entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming 13(3-4), 245–286 (1995)
11. Muggleton, S.: Inverse Entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming 13(3-4), 245–286 (1995)
12. Aleph, http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/
13. 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)
14. Fonseca, N.A., Rocha, R., Camacho, R., Silva, F.: Efficient Data Structures for Inductive Logic Programming. In: Horváth, T., Yamamoto, A. (eds.) ILP 2003. LNCS (LNAI), vol. 2835, pp. 130–145. Springer, Heidelberg (2003)
15. Nijssen, S., Kok, J.N.: Faster Association Rules for Multiple Relations. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), pp. 891–896 (2001)
16. Dehaspe, L., De Raedt, L.: Mining Association Rules in Multiple Relations. In: Džeroski, S., Lavrač, N. (eds.) ILP 1997. LNCS, vol. 1297, pp. 125–132. Springer, Heidelberg (1997)
17. Fredkin, E.: Trie Memory. Communications of the ACM 3, 490–499 (1962)
18. Bachmair, L., Chen, T., Ramakrishnan, I.V.: Associative-Commutative Discrimination Nets. In: Gaudel, M.-C., Jouannaud, J.-P. (eds.) CAAP 1993, FASE 1993, and TAPSOFT 1993. LNCS, vol. 668, pp. 61–74. Springer, Heidelberg (1993)
19. Graf, P.: Term Indexing. In: Graf, P. (ed.) Term Indexing. LNCS, vol. 1053, Springer, Heidelberg (1996)
20. Ramakrishnan, I.V., Rao, P., Sagonas, K., Swift, T., Warren, D.S.: Efficient Access Mechanisms for Tabled Logic Programs. Journal of Logic Programming 38(1), 31–54 (1999)
21. Yamamoto, A.: Which Hypotheses Can Be Found with Inverse Entailment? In: Džeroski, S., Lavrač, N. (eds.) ILP 1997. LNCS, vol. 1297, pp. 296–308. Springer, Heidelberg (1997)

Learning Relational Options for Inductive Transfer in Relational Reinforcement Learning

Tom Croonenborghs, Kurt Driessens, and Maurice Bruynooghe

K.U.Leuven, Dept. of Computer Science, Celestijnenlaan 200A, B-3001 Leuven

Abstract. In reinforcement learning problems, an agent has the task of learning a good or optimal strategy from interaction with his environment. At the start of the learning task, the agent usually has very little information. Therefore, when faced with complex problems that have a large state space, learning a good strategy might be infeasible or too slow to work in practice. One way to overcome this problem, is the use of guidance to supply the agent with traces of "reasonable policies". However, in a lot of cases it will be hard for the user to supply such a policy. In this paper, we will investigate the use of transfer learning in Relational Reinforcement Learning. The goal of transfer learning is to accelerate learning on a target task after training on a different, but related, source task. More specifically, we introduce an extension of the options framework to the relational setting and show how one can learn skills that can be transferred across similar, but different domains. We present experiments showing the possible benefits of using relational options for transfer learning.

Keywords: Relational Reinforcement Learning, Transfer Learning, Options.

1 Introduction

In reinforcement learning [12], an agent can observe its world and perform actions in it. The agent's learning task is to maximize the reward he obtains. In order to enable agents to learn in larger and more complex problem domains, abstraction has been an important factor. One important class of abstractions consists of hierarchical methods which focus on abstraction over the sequential and temporal aspects of a task [1]. Another direction of abstraction is relational reinforcement learning [14] which focuses on using relational representations for both the world (i.e., states and actions) and the learned policies.

One of the difficulties that remain is that at the start of the learning task, the agent has no or little information and is forced to perform random exploration. As a consequence, learning can become infeasible or too slow in practice for complex domains.

One of the approaches to tackle this problem is the integration of guidance in reinforcement learning [5]. In this approach, traces of "reasonable policies" are used to overcome the problem of forced random exploration. The drawback of this approach is that the user still needs to provide such a "reasonable policy" that will provide the learning agent with some positive reinforcement.

Another approach that targets this problem and which has received a lot of attention recently is inductive transfer or transfer learning. Transfer learning is concerned with learning in one task to benefit learning in different but related tasks. More specifically,

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 88–97, 2008.

in a reinforcement learning context, one of the benefits of transfer learning can be that the agent is able to learn a new task faster, i.e., with less training experience.

In this paper, we want to investigate the feasibility and benefit of using *hierarchical relational abstractions* for inductive transfer in reinforcement learning. The options framework is used for hierarchical abstractions [13]. Options are macro-actions that execute until some termination condition is satisfied and have been shown to be well suited to build high-level skills [9]. We introduce relational options as a combination of options with relational abstractions and use them to model skills that can be transferred across similar, but different domains.

Our contribution is threefold. First, we extend the options framework to the relational setting and show the benefits of such relational options. Second, we propose a method to learn options that can be used to transfer knowledge across different reinforcement learning domains using the framework of relational options. Third, we empirically evaluate if skills learned as options in previous tasks can help the reinforcement learning agent in more difficult tasks.

2 Background

2.1 Relational Reinforcement Learning and the Blocks World

Reinforcement Learning (RL) [12] is often formulated in the formalism of *Markov Decision Processes* (MDPs). An MDP $< S, A, T, R >$ can be characterized by a state space S, an action space A, a state transition function T and an immediate reward function R. The transition function $T : S \times A \times S \rightarrow [0, 1]$ defines a probability distribution over the possible next states: $T(s, a, s')$ denotes the probability of landing in state s' when executing action a in state s. The reward function $R : S \times A \rightarrow \mathbb{R}$ defines the reward for executing a certain action in a certain state.

The task of reinforcement learning consists of finding an *optimal policy* for a certain MDP, which is (initially) unknown to the RL-agent. As usual, we define it as a function of the discounted, cumulative reward, i.e. find a policy $\pi : S \times A \mapsto [0, 1]$, that maximizes the value function: $V^{\pi}(s) = \sum_{a \in A} \pi(s, a)[R(s, a) + \gamma \sum_{s'} T(s, a, s')V^{\pi}(s')]$, where $0 \leq \gamma < 1$ is the *discount factor*, which indicates the relative importance of future rewards with respect to immediate rewards.

The RRL-system [6] applies Q-Learning in relational domains, by using a relational regression algorithm to learn a Q-function that approximates the quality of executing a certain action in a certain state. The quality value for executing action a in state s with reward $R(s, a)$ and resulting state s' is defined as $Q(s, a) \equiv R(s, a) + \gamma\, max_{a'} Q(s', a')$. Knowing these Q-values, an optimal policy π^* can be constructed as $\pi^*(s) = \text{argmax}_a\, Q(s, a)$.

Throughout the paper, we will use the blocks world as an example application. We use a blocks world with a varying number of blocks, where blocks can only be stacked neatly on top of each other and the table or floor is of infinite size, i.e., it is always clear and ready to store an extra block. Consider a set of blocks $\{b_1, b_2, \ldots b_n\}$, every block b_i stands either on the floor (denoted $on(b_i, floor)$) or on some other block b_j (denoted $on(b_i, b_j)$) and on every block b_i there is either exactly one other block or it

is clear (denoted $clear(b_i)$). The agent can take a clear block b_i and put it on another clear block b_j or on the floor (denoted $move(b_i, b_j)$ or $move(b_i, floor)$ respectively).

2.2 The Options Framework

The theory of options has been introduced by Sutton et al. [13]. An option can be viewed as a subroutine, consisting of an option policy that specifies which action to execute for a subset of the environment states, an initiation set consisting of all states in which the option can be initiated and a termination condition, specifying when the option terminates. Note that an option is not just a sequence of actions, but a closed-loop policy taking actions depending on changes in the world.

More formally, an option o consists of three parts: 1) $\mathcal{I}_o \subseteq S : S \mapsto \{0, 1\}$ 2) $\beta_o : S \mapsto [0, 1]$ 3) $\pi_o : \mathcal{I}_o \times A \mapsto [0, 1]$ with $\mathcal{I}_o$ the initiate set which specifies the states in which the option can be initiated, β_o the termination condition which specifies the probability of terminating in state s for all $s \in S$ and π_o the *option policy* , which specifies the probability of executing a in state s for all $a \in A, s \in \mathcal{I}_o$. Since an option policy can also invoke other options, creating hierarchical structures, the action space A is extended to be the set of all options and primitive actions.

To update the Q-value of an option o that is started in state s and terminated in state s', the following equation can be used (similar to the one for primitive actions): $Q(s, o) \equiv R(s, o) + \gamma^d max_{o'} Q(s', o')$ with d the duration of the option (i.e. the number of time steps between s and s'). In this work, we also use the actions executed by the option policy to update their Q-values.

2.3 Transfer Learning

One of the motivations for relational reinforcement learning is that by using parameterized goal and policy descriptions it allows learned results to be applied in similar, but different worlds. This transfer of knowledge is however limited to learning problems that exhibit the same structure, only differing in the identity of certain objects or the number of objects involved [4].

Recently, several approaches have been proposed to transfer knowledge between different propositional reinforcement learning tasks. Often, a user-defined mapping is used to relate the new task to the task for which a policy was already learned.

Some approaches use the learned knowledge to aid exploration in new (and usually more difficult) tasks. Examples of these are [10] and [8]. The most important drawback of these methods is that a lot of useful information is lost when only using the transferred knowledge for exploration.

Perkins and Precup [11] investigate the use of non-relational options to transfer knowledge. They however only study the scenario where the agent has to solve different tasks drawn from a given distribution in which the agent does not know which task he has to solve during a certain episode. The approach of [9] also uses options to transfer knowledge where the need for a mapping between different tasks is avoided by introducing a so-called “agent-space”. This space is generated by a feature set that is present and retains the same semantics across successive problem instances.

The first approach that uses the generalizing power of relational learners is [15] where a relational rule learner is used to generate advice to speed up reinforcement learning. This advice is incorporated into the new task by adding the information about Q-values as soft-constraints to the linear optimization problem that approximates the Q-function for the next task which means that the advice needs to be propositionalized again for the new task. All advice is added to one big optimization problem, while the hierarchical abstractions in our approach are a first step towards a more modular representation of the policy.

3 Relational Options

The options framework can easily be extended to the relational setting by using a relational state and action space. We represent a relational policy by a set of rules of the following form: $\mathcal{C}_{\mathcal{S}} : \mathcal{C}_{\mathcal{SA}} \rightarrow \mathcal{A}$, with $\mathcal{C}_{\mathcal{S}}$ a conjunction of tests that only involve the state space, $\mathcal{C}_{\mathcal{SA}}$ a conjunction of tests on the state-action space and $\mathcal{A}$ the action predicted by this rule where the arguments are set by $\mathcal{C}_{\mathcal{SA}}$. When a policy needs to determine which action to execute in a state, it searches for the top-most rule for which $\mathcal{C}_{\mathcal{S}}$ is satisfied and an action $\mathcal{A}$ exists such that $\mathcal{C}_{\mathcal{SA}}$ holds. Action $\mathcal{A}$ (with its arguments set according to $\mathcal{C}_{\mathcal{SA}}$) will be predicted by the policy. Note that a distinction between $\mathcal{C}_{\mathcal{S}}$ and $\mathcal{C}_{\mathcal{SA}}$ is not strictly necessary. The predicate $s/1$ binds its argument with the current state and the $goal\_$-predicates are used to query the goal information: the $goal\_on/2$ predicate succeeds if the current goal of the agent is $on(A, B)$ and it will bind A and B with its arguments. Consider as an example an optimal policy for the $on(A, B)$ goal:

$$\begin{cases} s(S), goal\_on(A,B), clear(S,A), clear(S,B) : true \rightarrow move(A,B) \\ s(S), goal\_on(A,B), clear(S,A) : above(S,X,B), clear(S,X) \rightarrow move(X, floor) \\ s(S), goal\_on(A,B), clear(S,B) : above(S,X,A), clear(S,X) \rightarrow move(X, floor) \\ s(S), goal\_on(A,B) : above(S,X,A), clear(S,X) \rightarrow move(X, floor) \end{cases}$$

Although this relational extension of options is straightforward, it offers some advantages over regular options. Not only can they deal with relational worlds, it is also easy to extend them to a parameterized setting by expressing the policy in terms of variables instead of referencing concrete objects and by making it rely on the structural aspects of the task. Another advantage is that by having parameterized options, it also becomes possible to have recursive calls within the option policy.

Consider as an example an option that clears a certain block, instead of defining or learning a different option for every block that occurs in the world, it is possible to define one generalized option that can clear any block by introducing a variable as parameter of the option (as illustrated in Example 1). Note that this option does not specify a sequence of actions but a closed-loop policy deciding every time step which action to execute as long as the option is not terminated.

Example 1 (option for $clear(X)$).

$$\mathcal{I}_{clear(X)} : s(S) \mapsto 1$$

$$\beta_{clear(X)} : \begin{cases} s(S), clear(S,X) \mapsto 1 \\ s(S), \neg clear(S,X) \mapsto 0 \end{cases}$$

$$\pi_{clear(X)} : \begin{cases} s(S), goal\_clear(X) : on(S,Y,X), clear(S,Y) \rightarrow move(Y, floor) \\ s(S), goal\_clear(X) : on(S,Y,X) \rightarrow clear(Y) \end{cases}$$

4 Relational Skill Learning

One of the main difficulties in a lot of traditional transfer learning approaches is that in order to transfer knowledge one needs to provide a mapping from the source domain to the target domain. This problem is already partially overcome by using a relational representation since this allows us to abstract over specific object identities and even the number of objects involved.

In this paper we would like to transfer knowledge in the form of skills. Since we represent these skills as relational options, we need to specify the initiation set, the termination condition and the policy. To learn a specific option, the most straightforward approach would be to set the initiation set to $true$, the termination condition such that it is satisfied when a certain goal is reached and learn the policy with a standard relational reinforcement learning algorithm.

We take a slightly more advanced technique, motivated by the following observations: First of all, we would like to extract different skills from a single learning experience. Secondly, note that the Q-function contains more information than is actually needed for the optimal policy, i.e. it only requires a mapping from a state to the best action. Since the Q-function in a sense models the distance to reward, it does not always give the best generalization to new domains (with e.g. a different number of objects). Moreover, we would like the policy of the learned skill to be as interpretable as possible. Another disadvantage of traditional Q-learning approaches is that when an action needs to be predicted for a given state, an iteration is needed over all possible state-action pairs for that state to predict the action that results in the highest Q-value in case of a greedy policy. Since in complex domains with a lot of objects, the number of possible actions can be very large, we would like to avoid this iteration over all possible actions.

Therefore, we propose the following approach: The initiation set and termination condition can in the simplest case be set as specified above. Since the learned policy will not always be optimal it could be the case that the option never terminates. To avoid this scenario, the termination condition is changed so that every non-goal state has a non-zero probability of terminating the option. A skill policy for a certain task is built as follows where the first three steps are similar to the P-learning approach [6]:

1. use an RRL algorithm to learn to solve this task
2. create training examples of state-action pairs labeled as policy-based or not
3. use the TILDE system [2] to learn a relational decision tree that predicts whether or not the action will be executed by the policy
4. extract a relational policy

To create the dataset for TILDE, we create a number of random states and check which action would be executed by the learned policy. This action gets the label "policy action", other actions the label "non-policy action". For the moment we do not use any sampling techniques, but in the future we would like to investigate possible sampling techniques as well as different schema to create learning examples. One possibility is to only include examples of state-action pairs for which the Q-value is significantly better or worse than other state-action pairs for the same state (similar to the approach taken in [15]).

The learned decision tree for this binary classification problem predicts if, given a certain state and action, the action is the one that would be executed by the policy (or the

probability if we use probability trees). Although in future work we plan to investigate extraction schemas that obtain stochastic policies from these probability trees, at the moment we only look at the majority class in each of the leaves. This means we can do some bottom-up post-pruning if both the 'yes' and 'no' branch predict the same majority class. Figure 1 shows an example of a policy tree for the $clear(X)$ skill.

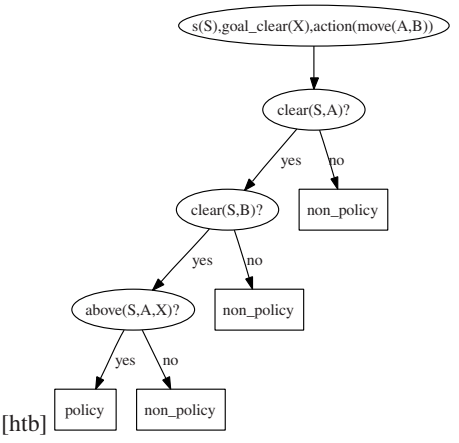


Fig. 1. Policy tree for $clear(X)$ that expresses that an action is a "policy action" if both arguments of the move action are clear and the agent moves a block above the one that needs to be cleared

To avoid the action iteration, we extract from this tree only those rules that predict the 'policy action' class. These rules give the constraints on the action to be part of this policy. If we do this for the tree in Figure 1, we only obtain the following rule: $s(S)$, $goal\_clear(X), action(move(A, B)), clear(S, A), clear(S, B),\ \ above(S, A, X) \rightarrow policy$. A policy is obtained by transforming each rule to the form $\mathcal{C}_{\mathcal{S}} : \mathcal{C}_{\mathcal{SA}} \rightarrow \mathcal{A}$, the order of the rules is not important since the decision tree partitions the state-action space. For computational reasons it could be interesting to first consider rules with the least negations in them. A default rule is also added that predicts a random action in case none of the other rules apply. For the policy tree in Figure 1, this gives the following relational policy:

$$\begin{cases} s(S), goal\_clear(X) : clear(S, A), clear(S, B), above(S, A, X) \rightarrow move(A, B) \\ s(S), goal\_clear(X) : random\_block(X), random\_block(Y) \rightarrow move(A, B) \end{cases}$$

There are different approaches to extract more than one skill. One possibility is to repeat the above procedure with different language bias settings, e.g. to obtain a different level of generalization. For instance the approach of [9] can be modeled with our approach by using appropriate language bias settings for both "problem-space" and "agent-space" skills. Another approach would be to specialize the initiation set of an option. For instance if the top test of the policy tree is not selective on the arguments of the action, the policy will be different for the two partitions of the state space.

5 Experimental Evaluation

Blocks World. In a first experiment, we consider the blocks world with as target task the $on(A, B)$-goal, i.e. the agent receives a reward iff block A is directly on top of block B. The number of blocks is varied between 5 and 15 every episode. During exploration, the learning agent is allowed 10 steps more than needed to reach the goal. To evaluate the agents' learning behavior, 100 tests are performed following a greedy policy checking the percentage of episodes in which an optimal path is followed. The RRL-TG[3] method is used in all experiments to approximate the Q-function. Figure 2(left) shows the results averaged over 5 different runs.

First of all, we will compare the learning behavior of a standard RRL agent using only primitive actions and an agent that can also use the $clear(X)$ option as defined in Example 1. As expected, Figure 2(left) clearly shows an improvement in learning behavior for the agent that can use this skill.

Next, we would like to investigate how difficult it is to learn such a $clear(X)$ skill. To learn this skill, we set up a blocks world environment with the $clear(X)$-goal in which the number of blocks is varied between 5 and 10. The Q-function learned by the agent after 100 episodes is taken to create training examples. This dataset is created using states from 50 new episodes[1]. The policy trees learned in the 5 different runs all looked similar to the one shown in Figure 1. They only differed in the order of the tests and the distributions in the leaves. Note that by only looking at the majority class noise in the Q-function can be filtered out. As a result the learned policy tree is optimal, while the policy based on the Q-function was not. Of course, by only looking at the majority class, it could also happen that important information is filtered out. We will address these issues in future work.

Since in general we do not know whether the learned skill is optimal, we set the termination condition so that it terminates for states in which $clear(X)$ holds and with a probability of 0.05 otherwise. Figure 2(left) shows the learning behavior of a learning agent using primitive actions and this learned option. Since an optimal policy is learned for this $clear(X)$ skill, the learning behavior is similar to the one with the user-defined $clear(X)$ skill and significantly better than the learning behavior of the agent with only primitive actions.

Tic-Tac-Toe. We also present an experiment using tic-tac-toe, an application from the general game playing challenge. If both players play optimally it results in a draw. The game is very asymmetric and although it is relatively easy to learn to draw against an optimal player when one is allowed to start the game, it becomes really hard to do this when the opposing player is allowed to act first. In fact, against a starting player that optimizes his game strategy against a random player, the probability of playing a draw with a random strategy is only 0.52%. This sparsity of the reward makes it very hard to build a good policy starting from scratch against an optimal player. We therefor devised an experiment which allowed the agent to first learn how to draw against a player that performs 1-step-look-ahead and transfer this knowledge as a skill to the experiment

[1] Instead of using episodes, it is also possible to create the dataset using random states, i.e. episodes of length one.

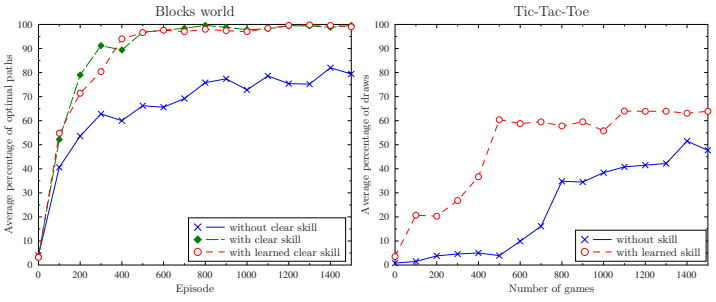


Fig. 2. Results in the blocks world (left) and the Tic-Tac-Toe (right) domain

against the optimal player. Note that the first is not a subtask of the latter. The 1-step-look-ahead player will play winning moves if they exist and counter winning moves of the opponent. However, in states where neither exist, it will play randomly and therefor will not be immune to the generation of forks. The skill is built in a similar way as in the blocks world experiment. To learn the option policy the agent plays 250 games against the 1-step-look-ahead player, states from 1000 random games are used to create the dataset. The language used by TG includes both non-game-specific knowledge (e.g. search related features) and game-specific features that can be automatically extracted from the specification of the game.

Figure 2(right) shows the improvement in learning behavior against the optimal player. In future work, we will investigate different possibilities to refine the termination condition of the learned skills while learning in the new task. This will be especially helpful in settings where the skill does not try to solve a subtask since the skill policy may only be useful in parts of the state space of the new task.

Grid World. In a last experiment, the agent is placed in an environment consisting of a sequence of two-dimensional rooms. The rooms are connected with doors which the agent can only pass if he possesses a key of the same color as the door. The primitive actions available to the agent include four movement actions (up,north,left and right) and a pickup action that picks up the key that is located at the agent's location (if applicable). The agent can execute at most 500 actions per episode and receives a reward of 1 if he exits the last room and 0 otherwise. Instead of using a single MDP, the agent has to solve different instantiations of this environment. Each problem consists of one to five rooms where the dimensions of each room are varied between three and five. Each room contains one to three keys of possibly different colors where at least one of them matches the door through which he can leave the room. The state representation includes the dimensions of the different rooms, the locations and colors of the doors, the location and colors of the keys, the keys the agent possesses, the agent's location and the goal location. The language bias consists of tests that can query the features of the state and action space and relational information between locations.

The agent can transfer knowledge through two different skills. The first skill is the $pickup\_key(Color)$ skill which finds a key of color $Color$ in the current room and picks it up. The second skill is the $find\_door$ skill which navigates the agent to the exit

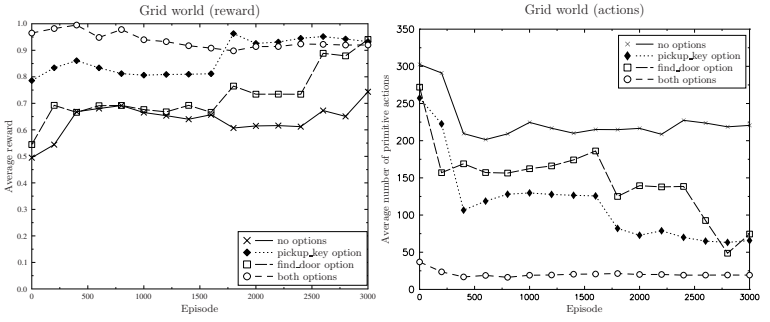


Fig. 3. Results in the multi-room grid wold domain

door in the current room. To learn the policies for these skills the agent is allowed 200 exploration episodes. The dataset is created using states from 100 different episodes.

To evaluate the performance of the different agents, we created 5 different sets, each containing 200 problem instantiations that were drawn randomly from the above distribution. For each of these sets the option policies were learned and we performed 5 different runs testing after every 200 exploration episodes the performance of a greedy policy on the entire set. The averages over these 25 runs are shown in Figure 3.

A first thing to note is that when the learning agent has extra skills to his disposal he can solve more problems using just random exploration. One can also see that an agent that learned the $find\_door$ option solves about the same number of problems as the standard RRL agent but he needs less actions on average. The agent with the $pickup\_key/1$ option solves more problems with less actions. If we draw problem instantiations from a distribution where finding the door is the bottleneck, only using the $find\_door$ option performs better than just using the $pickup\_key/1$ option[2]. An agent that has learned both skills always gives the best results.

6 Conclusions and Further Work

In this paper we presented an extension of the options framework to the relational setting. We have also shown how this framework can be used to learn relational skills that can be transferred across similar, but different tasks.

In future work, we will perform a more in depth analysis of the approach we presented. Currently, we have not considered the problem of determining which skills to learn. Since we learn parameterized options this is less of a problem, since it will often be possible to learn a skill for every predicate in the state representation (e.g. usually $clear/1$ and $on/2$ in the blocks world).

To decrease the distinction between the skill learning and exploitation phase or when learning these skills is hard, one could consider the approach of Fern et al. [7] to generate extra learning experience by using relational abstractions to create artificial goals in non-goal states. E.g. in every state there will be a clear block, so one can assign the clearance of that block as a goal to create training experience for the $clear(X)$ goal.

[2] We did not include the learning graphs of this experiment in the paper due to space restrictions.

Furthermore, at the moment we only consider the use of skills learned in previous tasks. A natural extension would be to focus on a hierarchical decomposition and learn new skills in the current task and to modify previous learned skills while learning in the new task.

Acknowledgments

This research is supported by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen) and by GOA 2003/08 "Inductive Knowledge Bases".

References

1. Barto, A., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. Discrete-Event Systems journal 13, 41–77 (2003)
2. Blockeel, H., De Raedt, L.: Top-down induction of first order logical decision trees. Artificial Intelligence 101(1-2), 285–297 (1998)
3. Driessens, K., Ramon, J., Blockeel, H.: Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. In: Flach, P.A., De Raedt, L. (eds.) ECML 2001. LNCS (LNAI), vol. 2167, pp. 97–108. Springer, Heidelberg (2001)
4. Driessens, K., Ramon, J., Croonenborghs, T.: Transfer learning for reinforcement learning through goal and policy parameterization. In: ICML Workshop on Structural Knowledge Transfer for Machine Learning (Online Proceedings) (2006)
5. Driessens, K., Dzeroski, S.: Integrating guidance into relational reinforcement learning. Machine Learning 57(3), 271–304 (2004)
6. Džeroski, S., De Raedt, L., Driessens, K.: Relational reinforcement learning. Machine Learning 43, 7–52 (2001)
7. Fern, A., Yoon, S., Givan, R.: Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. Journal of Artificial Intelligence Research 25, 85–118 (2006)
8. Fernández, F., Veloso, M.: Probabilistic policy reuse in a reinforcement learning agent. In: AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, pp. 720–727. ACM Press, New York (2006)
9. Konidaris, G., Barto, A.: Building Portable Options: Skill Transfer in Reinforcement Learning. In: Veloso, M. (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January, 6-12 2007, pp. 2895–900 (2007)
10. Madden, M.G., Howley, T.: Transfer of Experience Between Reinforcement Learning Environments with Progressive Difficulty. AI. Rev. 21(3-4), 375–398 (2004)
11. Perkins, T.J., Precup, D.: Using Options for Knowledge Transfer in Reinforcement Learning. In: Technical Report UM-CS-1999-034, University of Massachusetts, MA, USA (1999)
12. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. The MIT Press, Cambridge, MA (1998)
13. Sutton, R., Precup, D., Singh, S.: Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence 112, 181–211 (1999)
14. Tadepalli, P., Givan, R., Driessens, K.: Relational reinforcement learning: An overview. In: Proceedings of the ICML 2004 Workshop on Relational Reinforcement Learning (2004)
15. Torrey, L., Shavlik, J., Walker, T., Maclin, R.: Skill acquisition via transfer learning and advice taking. In: Proceedings of the 17th European Conference on Machine Learning, pp. 425–436 (2006)

Empirical Comparison of "Hard" and "Soft" Label Propagation for Relational Classification

Aram Galstyan and Paul R. Cohen

USC Information Sciences Institute
Center for Research on Unexpected Events (CRUE)
Marina del Rey, CA, USA
{galstyan,cohen}@isi.edu

Abstract. In this paper we differentiate between *hard* and *soft* label propagation for classification of relational (networked) data. The latter method assigns probabilities or class-membership scores to data instances, then propagates these scores throughout the networked data, whereas the former works by explicitly propagating class labels at each iteration. We present a comparative empirical study of these methods applied to a relational binary classification task, and evaluate two approaches on both synthetic and real–world relational data. Our results indicate that while neither approach dominates the other over the entire range of input data parameters, there are some interesting and non–trivial tradeoffs between them.

1 Introduction

Many relational classification algorithms work by iteratively propagating information through relational graphs. The main idea behind iterative approaches is that "earlier" inferences or prior knowledge about data instances can be used to make "later" inferences about related entities. Examples include relaxation labeling for hypertext categorization [1], belief propagation for probabilistic relational models [2], relevance propagation models for information retrieval on the web [3], iterative label propagation [4,5], relational neighbor classifiers [6,7,8].

While there are various ways of propagating information through relational graphs, here we differentiate between two general approaches: In the first approach, hard class label assignments are made at each iteration step. We call this approach *label propagation*[1] (LP). The second approach, which we call *score propagation* (SP), propagates soft labels such as class membership scores or probabilities. To illustrate the difference between these approaches, assume that we want to find fraudulent transactions given a relational graph of transactions (such as in Figure 1) and some known fraudulent nodes. For each transaction, we could estimate its probability of being fraudulent by using information about its neighboring nodes. The SP algorithm propagates these probabilities throughout the system and then makes a final inference by projecting the probabilities onto class labels. The LP algorithm, on the other hand, estimates these probabilities at

[1] We note that sometimes the term "label propagation" is also used to describe soft–label propagation.

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 98–111, 2008.

the first step, finds the entities with the highest probability of being fraudulent, labels them as fraudulent, and then iterates this procedure.

We would like to emphasize that despite the term "hard label propagation", we will focus on comparing two algorithms with respect to their accuracy of ranking rather than explicit classification. For the ranking problem, the difference between two approaches can be explained as follows: The SP algorithm is analogous to a diffusion-like process on a network, where initially labeled nodes act as heat sources, and the rank of a node is determined by its *temperature*. The LP algorithm, on the other hand, is similar to a discrete epidemic model, where, starting from initially *infected* nodes, the epidemic spreads to other nodes, and a node's rank depends on how early in the epidemic process it was infected.

Intuitively, one could think that the LP algorithm described above would not perform as well as the soft label propagation, since it makes hard "commitments" that cannot be undone later when more information is propagated through the network. Our main finding is that this is not always the case. We present results of extensive experiments for a simple binary classification task, using both synthetic and real–world data. For the synthetic data, we empirically evaluate both algorithms for a wide range of input parameters, and find that LP is usually a better choice if the overlap between the classes is not strong. More interestingly, we find that even when the performances of two algorithms are similar in terms of their AUC (area under the curve) scores, they might have significantly different ROC (Receiver–Operator Characteristics) curves: Specifically, we observe that this difference can be significant for small false positive rates. The other important observation is that for certain data parameters the LP algorithm is much more robust to the presence of noise in the initial class label assignments. Thus, propagating hard labels instead of continuos scores might be a better choice if the data is noisy.

In addition to our experiments on synthetic data, we tested both algorithms on the CoRA data–set of hierarchically categorized computer science papers. We constructed a separate classification problem for each CoRA topic in the Machine Learning category. Despite certain differences between our results for the CoRA and synthetic data, we observe that the hard label propagation scheme is indeed more robust to noise for the majority of the topics considered. Our CoRA experiments also reproduce the different ROC behaviors for certain topics, although the difference was not as significant as in the case of the synthetic data.

The rest of the paper is organized as follows: in the next section we state the binary classification problem and describe the synthetic data used in our experiments. We introduce hard and soft label propagation algorithms in Section 3. Section 4 describes related work. The results of experiments on both synthetic and the CoRA data are presented in Sections 5 and 6, respectively. Concluding remarks are made in Section 7.

2 Problem Settings

Most relational classification techniques rely on both intrinsic and relational attributes of data for making inferences. For instance, if the task is to classify scientific papers into topics, both intrinsic features (e.g., frequency of certain keywords) and relational

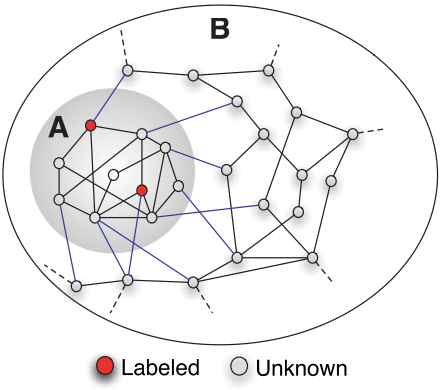


Fig. 1. Schematic representation of networked data

attributes (e.g., common authors, references, etc.) may be used. Here we are mainly interested in the relational aspect of the classification, so we ignore intrinsic attributes of the data instances and instead examine the effect of the relational structure on the classification accuracy. Specifically, we assume that the data is represented as an undirected graph, where nodes correspond to data instances and edges represent relationships between them.

Now we state the classification problem that we are interested in. Assume a relational graph as schematically illustrated in Figure 1. In the classification problem, one wants to find the set $\mathcal{A}$ of nodes that belong to class A (the shaded region), given the relational graph and a small subset $\mathcal{A}^0 \in \mathcal{A}$ of labeled A instances. In the ranking problem addressed in this paper, we are merely interested in ranking the nodes according to their similarity to the class A. We denote the nodes not in A as class B, and the corresponding set as $\mathcal{B}$. In general, class B itself might comprise of other classes that will be reflected in the topology of the network. This is the case for the CoRA data–set studied in Section 6. For the synthetic data, however, we will assume a homogenous structure for each class. Specifically, within each class, we randomly distribute links between pairs of nodes with probability $p_{in}^{a,b}$ so that the relational structures within the classes are characterized by Erdos–Renyi graphs $G(N_A; p_{in}^a)$ and $G(N_B; p_{in}^b)$[2]. N_A and N_B are the number of nodes in respective classes. Then we randomly establish links across the classes (blue edges in Fig. 1), by assigning a probability p_{out} to each of the $N_A N_B$ possible links. The average number of links per node (connectivities) within and across the classes are given by $z_{aa} = p_{in}^a N_A$, $z_{bb} = p_{in}^b N_B$, $z_{ab} = p_{out} N_B$ and $z_{ba} = p_{out} N_A$. If the sizes of two classes are not equal then $z_{ab} \neq z_{ba}$.

Note that our construction of the synthetic relational graph enforces the *homophily condition*: Namely, nodes from the same class are likely to be better-connected. Thus, we should expect the difficulty of the classification task to be strongly affected by the ratio of the connectivities within and across the classes. We will use the ratio $z_{ab}/z_{aa} \equiv z_{out}/z_{in}$ to characterize the degree of the homophily. A small ratio means that the

[2] Erdos–Renyi graph G(N; p) is constructed by independently linking each pair of N nodes with probability p.

classes are well–separated (strong homophily), so that most classification algorithms should do a good job of assigning correct class labels. For large values of z_{out}/z_{in}, on the other hand, the difference between the link patterns within and across the classes diminishes, thus making it more difficult to classify nodes correctly. We examine the effects of the class overlap on the classification accuracy in the experiments described in Section 5.

3 Algorithms

The score propagation mechanism employed here is very similar to the suspicion scoring model of Macskassy and Provost [9], as well as to the relevance propagation models from the information retrieval literature [3,10]. The label propagation algorithm, on the other hand, can be viewed a discrete (binary) analogue of the score propagation scheme. Below we describe both approaches in more details.

3.1 Score Propagation

By score propagation we mean a type of iterative procedure that propagates continuous–valued class membership scores from the labeled class instances to the unlabeled ones. In our case, the initially labeled set contains only nodes of type A. Hence, we associate a score s_i with node i, that describes its relative likelihood of being in the class A. These scores are updated iteratively, allowing the influence of the labeled nodes to spread throughout the data. The main assumption behind this scheme is that the nodes that belong to the class A will have higher scores at the end of the propagation process.

There are many possible ways to implement the propagation mechanism. Here we use a scheme described by the following equation:

$$s_i^{t+1} = s_i^0 + \alpha_i s_i^t + \beta \sum_j W_{ij} s_j^t, \tag{1}$$

where s_i^0 is a static contribution that might depend on a node's intrinsic attributes, α_i and β_i are parameters of the model, $W_{ij} = 1$ if nodes i and j are connected and $W_{ij} = 0$ otherwise. For instance, in the relevance propagation model from the information retrieval literature, s_i^0 is the content–based self-relevance score of node i, $\alpha_i = const < 1$, and $\beta_i = (1-\alpha_i)/z_i$, where z_i is the number of neighbors of node i. In the suspicion scoring model of Ref. [9], $s_i^0 = 0$, $\alpha_i = \alpha$ for all i, and $\beta = (1-\alpha)/\sum_{i,j} W_{ij}$.

Our experiments with variants of the SP schemes suggest that they all behave in qualitatively similar ways. Here we report results for a simple parameter-free version obtained by setting $s_i^0 = \alpha_i = 0$, and $\beta_i = 1/z_i$. The resulting updating scheme is

$$s_i^{t+1} = \frac{1}{z_i} \sum_j W_{ij} s_j^t. \tag{2}$$

In other words, at each iteration, the class membership score of a node is set to the average of the class–membership scores of its neighbors at the previous iteration. We note that this model closely resembles the random walk model of Ref. [11].

The scores of the initially labeled A nodes are *clamped* to 1, while the rest of the nodes are initially assigned a score 0. Because of the clamping, the former nodes act as diffusion sources, so that the average score in the system increases with time and in fact converges to 1. Therefore, we stop the iteration after the average score exceeds some predefined threshold, chosen to be 0.9 in the experiments reported below. We observed that the final ranking of the nodes according is not sensitive to the choice of this threshold.

3.2 Label Propagation

For the hard label propagation, LP, we propose a simple mechanism that is in some sense the discrete (binary) analogue of the SP scheme. Let us assign binary state variables $\sigma_i = \{0, 1\}$ to all nodes so that $\sigma_i = 1$ (or $\sigma_i = 0$) means that the i–th node is labeled as type A (or is unlabeled). At each iteration, and for each unlabeled node, we calculate the fraction of the labeled nodes among its neighbors, $\omega_i^t = \sum_j W_{ij}\sigma_j^t/z_i$, find the nodes for which the fraction is the highest, and label them as type A . This procedure is then repeated for T_{max} steps.

The label propagation process above can be viewed as a combination of the score propagation scheme from the previous section and a nonlinear transformation applied to the scores after each iteration. This nonlinear transformation constitutes a simple inference process where the class-membership scores of a subset of nodes are projected into class labels. Indeed, starting from the initially labeled instances, let us iterate the SP scheme in Equation 2 once. Then, obviously, on has $s_i^1 = \omega_i^1$. That is, the nodes that have the largest fraction of the labeled nodes among their neighbors, also have the highest score. The step-like transformation then assigns a score 1 to all the nodes sharing the maximum score, and sets the score of the remaining nodes to zero, thus acting as a filter.

While ranking nodes in the SP scheme is straightforward, we need a different ranking mechanism for the LP scheme. Note that the only parameter of the LP classification scheme is the number of iterations T_{max}. In particular, by choosing different T_{max} one effectively controls the number of labeled instances. Hence, setting T_{max} is in a sense analogous to setting a classification threshold for the SP mechanism. This suggests the following natural criterion for ranking: Rank the nodes according to the iteration time step when they were labeled as type A, so that a node that is labeled earlier in the iteration has a lower rank (i.e., is more likely to belong to the class A). The justification of this approach is again based on the homophily condition: nodes that are similar to the initially labeled nodes will tend to be better connected with them, hence they will be labeled earlier in the iteration.

4 Related Work

Before presenting our experimental results, we would like to clarify the connection of the models in Section 3 with some existing work. The score propagation model in Equation 2 is a special case of the suspicion scoring model of Macskassy and Provost [9]. One subtle difference is that Ref. [9] uses annealing to guarantee convergence, by

decreasing α with time. Another aspect of the work in [9] is an adaptive data access based on iterative runs of the scoring scheme. Specifically, after a first run of the SP scheme, they choose the top K nodes, query them against a secondary database, and augment the network with new links. Then they run the SP scheme again to generate new rankings. Since in our model the relational graph is given initially, we do not perform iterations over many SP schemes. We note, however, that our LP algorithm is analogous to performing multiple iterations over the score propagation scheme, where each SP run includes only one iteration of the Equation 2.

Recently there has been a growing interest in the information retrieval community in using both link and content information for web queries [12]. The SP model is strongly related to the relevance propagation models for the web-based information retrieval [3,10]. One of the differences is that our model does not have the self–relevance term that describes a node's content. Also, the graph in our model is undirected, while in the web mining applications the link directionality plays an important role (see also [13]).

The classification problem considered here is related to semi-supervised learning with partially labeled data. Recently, several algorithms that combine both labeled and unlabeled data have been suggested [11,14,15]. Remarkably, these approaches too are based on the assumption that nearby data points are likely to belong to the same class. Given a dataset with partially labeled examples, Zhu and Ghahramani [15] construct a fully connected graph so that the weight of the link between two data points x_1 and x_2 depends on the distance $d(x_1, x_2)$. They then suggest a "soft" label propagation scheme where the information about the labeled nodes is propagated throughout the constructed graph. Thanks to the problem formulation, they were able to avoid the actual propagation step and instead solve a linear system of equations to obtain the class–membership probabilities. Despite obvious similarities, there are also important differences with the model considered here. First of all, the scores in our model are not interpretable as probabilities. Also, the algorithm in Ref [15] works only if there are initially labeled data points from both classes (for binary classification), while in our case we do not have that constraint, and need only positive examples.

5 Experiments with Synthetic Data

We evaluated the algorithms using ROC analysis, and particularly, AUC (Area Under the Curve) scores. In our experiments with synthetic data, we used equal class sizes $N_A = N_B = 500$ in one of the experiments, and skewed class distribution with $N_A = 200$ and $N_B = 2000$ in all the others. We run 100 trials for each choice of the parameters, and calculated both the average and the standard deviation of the AUC scores over the trials.

5.1 Class Overlap

In the first set of experiments, we examine the effect of the class overlap on the classification accuracy. As we already mentioned, the class overlap can be measured by the ratio z_{out}/z_{in}. In Figure 2 we plot the AUC score against the ratio z_{out}/z_{in} for

three different values of z_{in}. The top panel shows the results for equal class sizes, $N_A = N_B = 500$, with the number of initially labeled instances $N_A^0 = 100$, e.g., 20% of all A nodes. Starting from a near-perfect AUC scores at the ratio 0.1 for $z_{in} = 5$, the accuracies of both SP and LP degrade gradually with increasing the ratio z_{out}/z_{in}. Specifically, for $z_{in} \approx z_{out}$ the AUC scores falls to 0.5, which corresponds to randomly generated ranking. We also note that there is a crossover region in the performances of the algorithms: at $z_{out}/z_{in} = 0.1$, LP attains slightly higher AUC score than SP, while for $z_{out}/z_{in} \geq 0.5$ the SP algorithm performs better. This pattern is amplified for larger within–class connectivities. Indeed, for $z_{in} = 20$ both algorithms attain near–perfect AUC scores for ratios up to 0.3, and for $z_{out}/z_{in} > 0.3$, LP clearly outperforms SP up until the crossover point at 0.7, with the difference in the AUC scores as high as 0.1 at certain points. More interestingly, the crossover point shifts to right with increasing the within-class link density. This suggests that for sufficiently dense graphs, LP is a better choice if the class overlap is not very large. For sparse graphs and relatively large overlap, however, SP performs better.

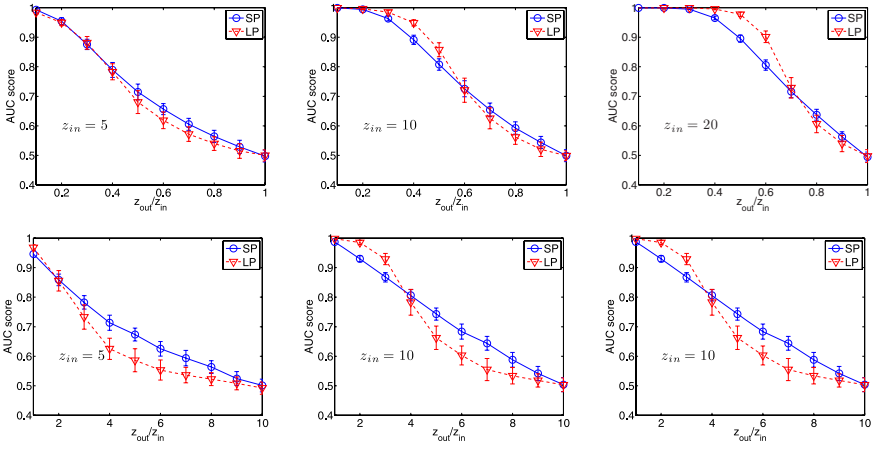


Fig. 2. AUC score vs the ratio z_{out}/z_{in} for different values of z_{in}. The top and bottom panels are for equal and skewed class distributions respectively.

A similar picture holds in the presence of a class skew (bottom panel in Fig. 2). The number of nodes in each class is $N_A = 200$ and $N_B = 2000$, with 20% of A nodes initially labeled (i.e., $N_A^0 = 40$). The only difference from the previous case is that the ratio at which the performances of both algorithms fall to the random level is now shifted towards the higher values of z_{out} (note that the horizontal axis ranges from 1 to 10). The reason for this is as follows: For a given z_{out}, each type B node is connected with $z_{ba} = z_{out}N_A/N_B$ type A nodes in average. One should expect the ranking to be random when a B node has roughly the same number of A neighbors as an A node. Thus, the corresponding z_{out} can be estimated from $z_{out}/z_{in} \sim N_B/N_A$. For the class skew of 10, this estimate yields $z_{out}/z_{in} \sim 10$, which agrees well with the experiment.

5.2 ROC Analysis

We now present the ROC analysis of the algorithms, for a fixed within–class connectivity $z_{in} = 5$, and three different choices of the class overlap $z_{out} = \{5, 10, 15\}$. For $z_{out} = 5$, LP achieves a slightly better AUC score than SP. For $z_{out} = 10$ both algorithms have the same AUC score (within the standard deviation). And finally, for $z_{out} = 15$ SP has a better AUC score (see the bottom panel inFig. 2). In the experiments, we used bins of size 0.01 for the false positive rate (FP), and for each bin we calculated the average and standard deviation of the corresponding true positive rate (TP). The results are shown in Fig. 3.

We first discuss the case $z_{out} = 5$. The corresponding AUC scores are 0.95 ± 0.01 for SP and 0.97 ± 0.01 for LP. What is remarkable, however, is that despite this tiny difference, the two classifiers are quite distinct for small false positive rates. In other words, the difference in the AUC scores is not distributed equally over the whole ROC plane. Instead, the main difference is for the false positive range $0 < FP < 0.1$. For $FP > 0.3$, on the other hand, SP achieves marginally better true positive rates. This observation suggests that if the cost of false positives are high, then LP is a superior choice for small class overlap. This can be especially important in the case of a highly skewed class distribution, where even tiny false positive rates will translate into a large number of falsely classified instances. The inset shows the difference between true positive rates, $\Delta TP = TP_{LP} - TP_{SP}$, as a function of FP. The bars in the plot are two

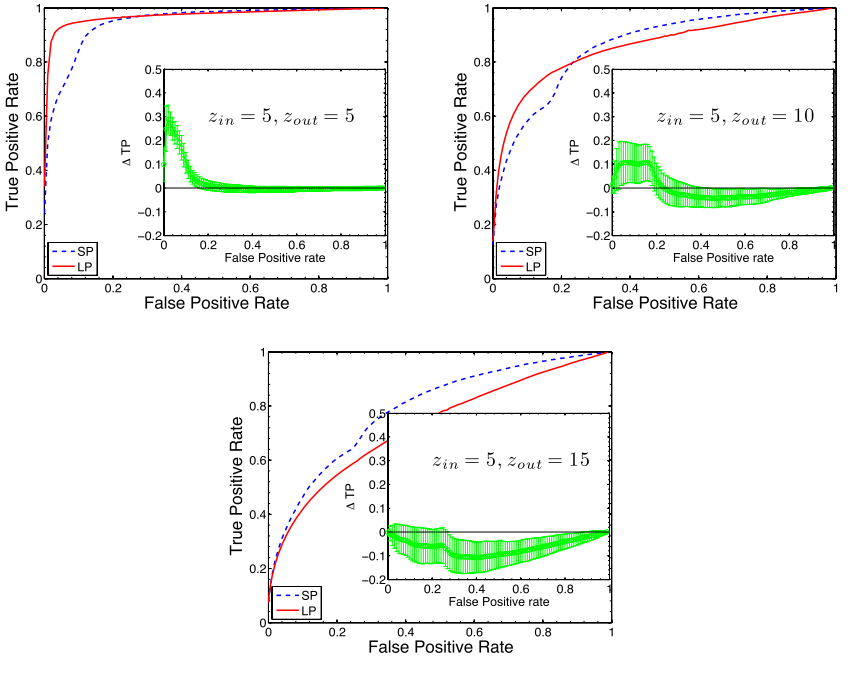


Fig. 3. ROC curves for different connectivities

standard deviations wide and centered around the mean. Clearly, for a small interval around $FP = 0.05$, this difference is positive and statistically significant, and achieves a value as high as ~ 0.3.

A somewhat similar, although less dramatic, effect holds for $z_{out} = 10$, when the AUC scores of both algorithms are virtually indistinguishable. In this case, LP achieves better true positive rates in the range $0 < FP < 0.2$, while SP performs better on the rest of the axis. The difference between them is not as pronounced as in the previous case, and the corresponding standard deviations are higher. Finally, for $z_{out} = 15$ the SP algorithm matches the performance of LP for small positive rates, and outperforms the latter over the rest of the FP axis. This again suggests that for relatively large class overlap SP is a better choice. Followup experiments revealed that the observed differences in ROC curves, especially for small false positive rates, persist for a wide range of parameters, as long as the overlap between the classes is not very large. Moreover, the difference becomes more dramatic for larger within-class connectivities z_{in}. For some parameters this difference was as high as 0.5 for small FP rates.

5.3 Effect of Noise

Next, we study how the classification accuracy deteriorates in the presence of noise, which was introduced by randomly and uniformly choosing N_B^0 nodes from the class B and mislabeling them as type A initially. In the experiments, we set the number of initially labeled A nodes to $N_A^0 = 40$, and studied how the AUC score changed as we increased the number of mislabeled nodes, N_B^0. The results are presented in Fig. 4, where we plot the AUC score against the ratio N_B^0/N_A^0 for three different values of the class overlap. Remarkably, for a small class overlap, $z_{out} = 10$, the noise has distinctly different effects on SP and LP. The LP algorithm seems to be very resilient to the noise and has an AUC score close to ~ 0.97 even when the number of mislabeled nodes is $N_B^0 = 200$, or five times the number of correctly labeled nodes. The performance of the SP algorithm, on the other hand, deteriorates steadily starting from moderate values of noise and attains an AUC score of only 0.68 for $N_B^0 = 200$. A similar, although weaker, effect is observed for moderate overlap $z_{out} = 20$. The AUC score of the SP algorithm decreases almost linearly, while for the LP algorithm the decrease is much slower. Finally, for $z_{out} = 30$ the noise seems to affect the performance of both algorithms very similarly.

6 Experiments with the CoRA Data

The assumption that the relational structure is described by coupled Erdos-Renyi graphs might not be appropriate for real world datasets. Hence, it is important to find out whether the results described in the previous sections hold for more realistic data. In this section we present our experiments on the CoRA data–set of hierarchically categorized computer science research papers [16]. We focus on the papers in the Machine Learning category, which contains seven different subtopics: “Case-Based”, “Genetic Algorithms”, “Probabilistic Methods”, “Neural Networks”, “Reinforcement Learning”, “Rule Learning” , and “Theory”. Two papers were linked together by using common

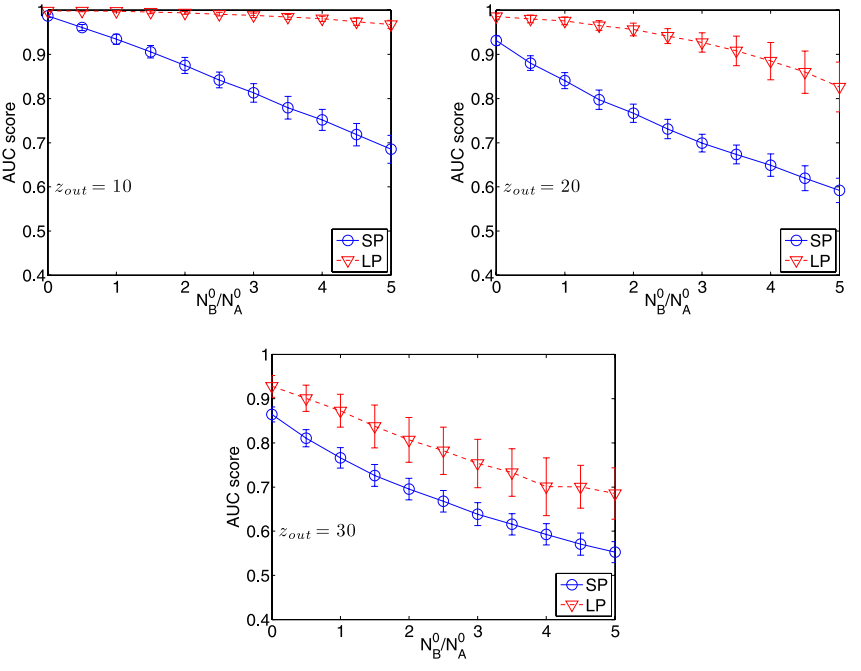


Fig. 4. The AUC score plotted against the ratio N_B^0/N_A^0. The within–class connectivity is $z_{in} = 10$.

author (or authors) and citation. After pruning out isolated papers from the data–set, we were left with 4025 unique titles. In our experiments, we mapped the multi–class problem onto a binary classification problem for each individual topic.

Generally speaking, the results obtained for the CoRA data were somewhat different from the results for the synthetic data. Specifically, we found that the ranking accuracies were lower than one would expect for a random Erdos–Renyi topology with corresponding connectivities, especially for the LP algorithm. We believe that this is due to the fact that the CoRA graph has a much more skewed degree distribution compared to the exponential distribution of Erdos–Renyi graphs (indeed, we established that the performances of both algorithms improve if we purge nodes with very high and very low connectivities from the graph). We also found that in contrast to the synthetic data, the SP algorithm was usually better than LP in case when there was no noise in the initial label assignment.

Despite these differences, however, we established that our main results for the synthetic data also hold for some of the CoRA topics. In particular, we observed that for four out of the seven topics the LP algorithm is indeed less sensitive to noise. This is shown in Figure 5 where we plot the AUC score against the fraction of the mislabeled nodes for six of the topics. For the topics "Genetic Algorithms", "Reinforcement Learning", "Rule Learning" , and "Theory", the decrease in the accuracy for the LP algorithm is smaller than for the SP algorithm, although the difference is not as dramatic as in

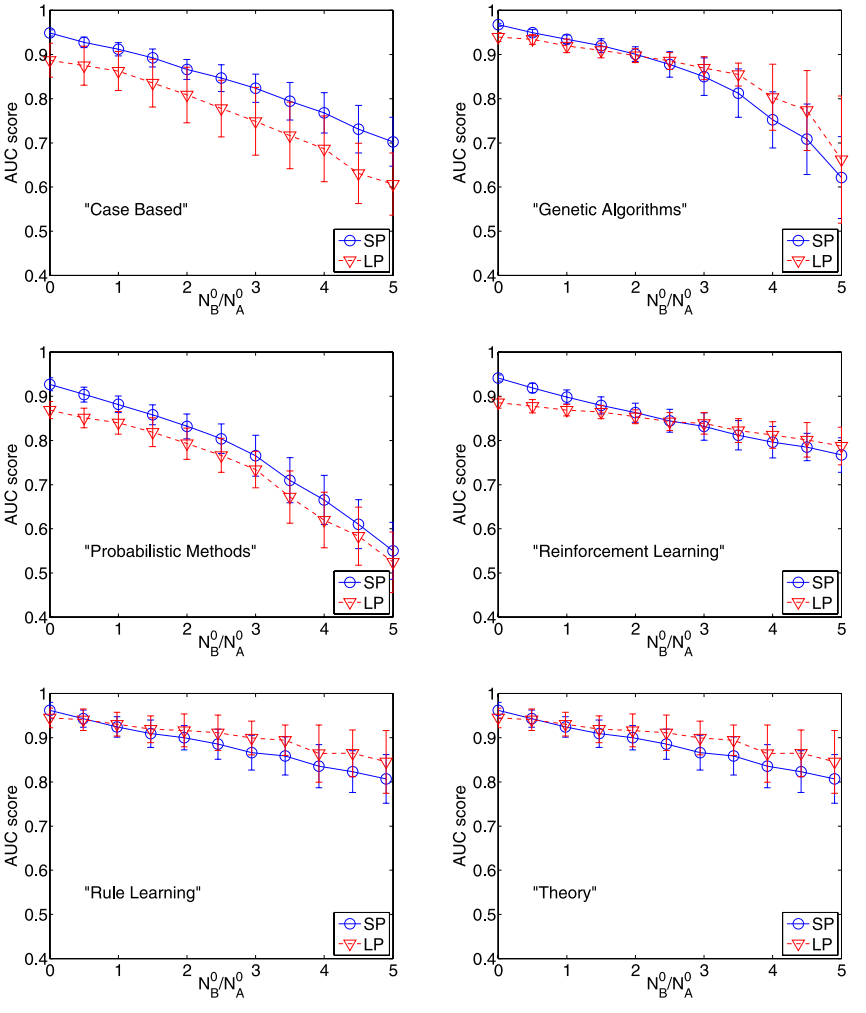


Fig. 5. The AUC score plotted against the ratio N_B^0/N_A^0 for different CoRA topics

the case of the synthetic data. For two other topics, "Case-Based" and "Probabilistic Methods", as well as for the "Neural Networks" topic not shown here, both algorithms responded similarly to the noise.

Furthermore, in Figure 6 we show the ROC curves for the same topics. Again, for some of the topics the observed picture is qualitatively very similar to that presented in Figure 3 for the synthetic data. Namely, although the AUC scores of both algorithms are very close, their ROC curves are different, with LP achieving better accuracy for smaller false positive rates. This is especially evident for the "Reinforcement Learning" topic for which the (average) maximum difference is close to 0.18. Note also that for

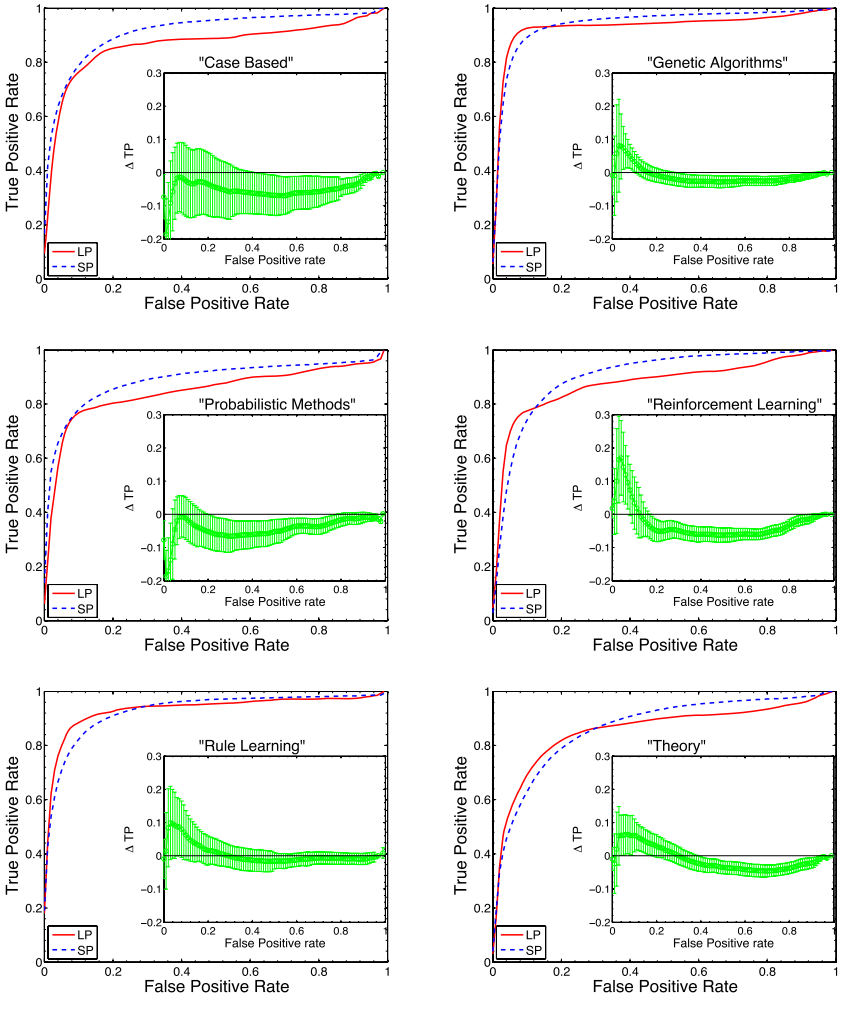


Fig. 6. ROC curve for different CoRA topics

the “Case-Based” and the “Probabilistic Methods” topics SP outperforms LP for the whole ROC plane (this is also true for the “Neural Networks” topic, not shown here).

7 Discussion and Future Work

We have presented empirical comparison of *hard* and *soft* label propagation techniques for binary classification in relational data. Our results suggest that for sufficiently strong homophily of the linked data, both methods achieve a remarkably good ranking accuracy. We also found that, while neither of the approaches dominates over the entire range

of parameters, there are some important differences that should be taken into account for deciding which one is better suited for a particular problem.

One of the main findings of our paper is that even when two algorithms achieve the same accuracy of ranking (as characterized by their AUC scores), the behavior of the family of classifiers based on them can be drastically different. Specifically, we found that for small values of allowed false positive rates, LP usually achieves higher true positive rates. In fact, for data with small class overlap, the observed difference was quite dramatic. The SP algorithm, on the other hand, achieves higher true positive rates for larger allowed false positive rate. This suggests that SP might be a better choice only when the cost of false negatives strongly outweighs the cost of false positives. This difference will be especially important in the case of highly skewed class distributions, where even tiny false positive rates translate into a large number of falsely classified instances.

The other important finding is the different behavior of the two propagation schemes in the presence of noise. Our experiments with synthetic data, as well as for some of the CoRA topics, suggest that the LP algorithm is more robust to the presence of mislabeled data instances. Thus, propagating hard labels instead of scores might be a better choice if the prior information is noisy. We believe that this is an interesting observation that warrants a further examination, both analytically and empirically.

We also note that the algorithms have different computational complexities. Indeed, the worst case time–complexity of the LP algorithm scales linearly with the number of data instances, as it might require N iterations to rank N instances. This correspond to the case when only one node is labeled at each iteration step. The SP algorithm, on the other hand, scales much better with the data size. In fact, our experiments show that the relative ranking almost saturates once the influence propagates from the seed nodes to the rest of the nodes, which happens after a much shorter time scale (order of $\sim \log N$). This difference can be very important for very large scale data.

Many relational classification techniques rely on information propagation over graphs. However, there are not many systematic studies that examine the role of the graph structure on the propagation dynamics. We have addressed this problem for fairly simple propagation dynamics and a graph topology. We believe it would be worthwhile to perform similar studies for more sophisticated classification schemes, and extend the empirical framework presented here to more complex relational domains. Currently, evaluations of various relational learning algorithms are limited to a handful of real world datasets. While it is important to perform well on real world data, we believe that evaluating algorithms through a controlled set of experiments on synthetic data will help to better understand their strengths and weaknesses.

References

1. Chakrabarti, S., Dom, B.E., Indyk, P.: Enhanced hypertext categorization using hyperlinks. In: Haas, L.M., Tiwary, A. (eds.) Proceedings of SIGMOD-1998, ACM International Conference on Management of Data, Seattle, US, pp. 307–318. ACM Press, New York (1998)
2. Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: Proceedings of the IJCAI-1999, pp. 1300–1309 (1999)

3. Qin, T., Liu, T.Y., Zhang, X.D., Chen, Z., Ma, W.Y.: A study of relevance propagation for web search. In: SIGIR 2005: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 408–415. ACM Press, New York (2005)
4. Galstyan, A., Cohen, P.R.: Inferring useful heuristics from the dynamics of iterative relational classifiers. In: Proceedings of IJCAI-2005, 19th International Joint Conference on Artificial Intelligence (2005)
5. Galstyan, A., Cohen, P.R.: Relational classification through three–state epidemic dynamics. In: Proceedings of the 9th International Conference on Information Fusion, Florence, Italy (2006)
6. Macskassy, S., Provost, F.: A simple relational classifier. In: Proceeding of the Workshop on Multi-Relational Data Mining in conjunction with KDD-2003 (MRDM-2003), Washington, DC (2003)
7. Macskassy, S., Provost, F.: Classification in networked data: A toolkit and a univariate case study. In: Working paper CeDER-04-08, Stern School of Business, New York University (2004)
8. Macskassy, S., Provost, F.: Netkit-srl: A toolkit for network learning and inference. In: Proceeding of the NAACSOS Conference (2005)
9. Macskassy, S., Provost, F.: Suspicion scoring based on guilt-by-association, collective inference, and focused data access. In: Proceeding of the International Conference on Intelligence Analysis, McLean, VA (2005)
10. Shakery, A., Zhai, C.: Relevance propagation for topic distillation uiuc trec 2003 web track experiments. In: TREC, pp. 673–677 (2003)
11. Szummer, M., Jaakkola, T.: Partially labeled classification with markov random walks. In: Advances in Neural Information Processing Systems, vol. 14 (2001)
12. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project (1998)
13. Gyongyi, Z., Garcia-Molina, H., Pedersen, J.: Combating web spam with trustrank. In: Proceedings of the 30th VLDB Conference (2004)
14. Tishby, N., Slonim, N.: Data clustering by markovian relaxation and the information bottleneck method. In: NIPS, pp. 640–646 (2000)
15. Zhu, X., Ghahramani, Z.: Learning from labeled and unlabeled data with label propagation. Technical report, Carnegie Mellon University (2002)
16. McCallum, A., Nigam, K., Rennie, J., Seymore, K.: Automating the construction of internet portals with machine learning. Information Retrieval Journal 3, 127–163 (2000)

A Phase Transition-Based Perspective on Multiple Instance Kernels

Romaric Gaudel[1,2], Michèle Sebag[1], and Antoine Cornuéjols[3]

[1] CNRS − INRIA − Univ. Paris-Sud, F-91405 Orsay, France
{romaric.gaudel, michele.sebag}@lri.fr
[2] École Normale Supérieure de Cachan
[3] AgroParisTech − INRA, F-75005 Paris, France
antoine.cornuejols@agroparistech.fr

Abstract. This paper is concerned with Relational Support Vector Machines, at the intersection of Support Vector Machines (SVM) and Inductive Logic Programming or Relational Learning. The so-called phase transition framework, originally developed for constraint satisfaction problems, has been extended to relational learning and it has provided relevant insights into the limitations and difficulties thereof. The goal of this paper is to examine relational SVMs and specifically Multiple Instance (MI) Kernels along the phase transition framework. A relaxation of the MI-SVM problem formalized as a linear programming problem (LPP) is defined and we show that the LPP satisfiability rate induces a lower bound on the MI-SVM generalization error. An extensive experimental study shows the existence of a critical region, where both LPP unsatisfiability and MI-SVM error rates are high. An interpretation for these results is proposed.

Keywords: Phase Transition, Multiple Instance Problems, Relational Learning, Relational Kernels, Support Vector Machines.

1 Introduction

This paper is concerned with Relational Support Vector Machines, at the intersection of Support Vector Machines (SVM) [20] and Inductive Logic Programming or Relational Learning [18]. After the so-called kernel trick, the extension of SVMs to relational representations relies on the design of specific kernels (see [8,10] among many others). Relational kernels thus achieve a particular type of propositionalization [14], mapping every relational example onto a propositional space defined after the training examples. However, relational representations intrinsically embed combinatorial issues; for instance the Plotkin's θ-subsumption test used as relational covering test is equivalent to a Constraint Satisfaction Problem (CSP) [11]. The fact that relational learning involves the resolution of CSPs as a core routine has far-fetched consequences besides exponential (worst-case) complexity, referred to as the Phase Transition (PT) paradigm (more on this in section 2).

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 112–121, 2008.

The question investigated in this paper is whether relational SVMs overcome the limitations of relational learners related to the PT [3]. Specifically, the study focuses on the Multiple Instance (MI) setting [9], for which several SVM approaches have been proposed [10,8,16,15]. This paper presents two contributions. Firstly, a relaxation of the MI-SVM problem is introduced and formalized as a Linear Programming Problem (LPP); we show that the LPP satisfiability rate derives a lower bound on the generalization error of the MI-SVM. Secondly, a principled experimental study is conducted, based on a set of order parameters; these experiments show the existence of a critical region, conditioned by the value of order parameters, where both LPP unsatisfiability and MI-SVM error rates are high.

The paper is organized as follows. For the sake of self-containedness, the Phase Transition framework is briefly introduced in Section 2 together with the Multiple Instance setting. Section 3 defines a relaxed formalization of the MI-SVM expressed as a LPP, and establishes a relation between the MI-SVM generalization error and the LPP satisfiability rate. Section 4 reports on the experimental study and discusses the results. The paper concludes with some perspectives for further research.

2 State of the Art

It is widely acknowledged that there is a huge gap between the empirical and the worst case complexity analysis for CSPs [4]. This remark led to developing the so-called *phase transition paradigm* (PT) [12], which considers the satisfiability and the resolution complexity of CSP instances as random variables depending on order parameters of the problem instance (e.g. constraint density and tightness).

The phase transition paradigm has been transported to relational machine learning and inductive logic programming (ILP) by [11], and was shown to be instrumental in discovering and analyzing some limitations of relational learning [3] or grammatical inference [19] algorithms, such as the existence of a failure region for existing relational learners [3].

Resuming the above studies, this paper investigates the PT phenomenon in the Multiple Instance Learning setting introduced by Dietterich et al. [9], which is viewed as intermediate between relational and propositional settings. Formally, a MI example $\mathbf{x}$ is a bag of (propositional) instances noted $x^{(1)}$, ..., $x^{(N)}$.

In the original MI setting, referred to in the following as *linear*, an example is labelled positive iff it includes at least one instance satisfying some target concept C:

$$pos(\mathbf{x}) \text{ iff } \exists\, i \in 1 \ldots N \ s.t.\ C(x^{(i)})$$

However, in some contexts such as image categorization, [5] pointed out that the example label might depend on the properties of several instances; along the same lines, several alternative formalizations were proposed by [21] and the remainder of the paper will consider the so-called *presence-based* setting, with:

$$pos(\mathbf{x}) \text{ iff } \forall\, j = 1 \ldots m,\ \exists\, i_j \in 1 \ldots N \ s.t.\ C_j(x^{(i_j)})$$

Many approaches have been developed to address MI problems, including specific algorithms focussing on *linear* MI [17,22], relational algorithms [6,2], and specific Support Vector Machine (SVM) approaches [10,8,16,15]. Assuming the reader's familiarity with SVMs [20] and restricting ourselves to standard bag kernels in this paper, MI-kernels K are constructed on the top of propositional kernels k. Formally, letting $\mathbf{x} = (x^{(1)}, \ldots x^{(N)})$ and $\mathbf{x}' = (x'^{(1)}, \ldots x'^{(N')})$ denote two examples, standard MI-kernels are defined as:

$$K(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}).f(\mathbf{x}') \sum_{k=1}^{N} \sum_{\ell=1}^{N'} k(x^{(k)}, x'^{(\ell)}) \tag{1}$$

where $f(\mathbf{x})$ corresponds to a normalization term, e.g. $f(\mathbf{x}) = 1$ or $1/N$ or $1/\sqrt{K(\mathbf{x}, \mathbf{x})}$.

MI-SVMs have obtained good results on *linear* MI problems [10], and also in application domains which rather belong to the *presence-based* setting, such as image categorization [15] or chemometry [16].

Still, by construction standard MI-kernels consider the average similarity among the example instances. The question examined in this paper is to which extent this average information is sufficient to reconstruct existential concepts involved in *presence-based* MI problems.

3 Overview

This section introduces a relaxation of MI-SVM problems in terms of Linear Programming problems, which will be exploited to analyze MI-SVM along the phase transition framework.

3.1 When MI Learning Meets Linear Programming

In order to investigate the performance of an algorithm within the PT framework, a standard procedure is to generate artificial problems after the selected order parameters (see below), where each problem is made of a training set $\mathcal{L} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$ and a test set $\mathcal{T} = \{(\mathbf{x'}_1, y'_1), \ldots, (\mathbf{x'}_t, y'_t)\}$, and to compute the error on the test set of the hypothesis learned from the training set. The test error, averaged over a sample of artificial problems generated after some order parameter values, indeed measures the competence of the algorithm conditionally to these parameter values [3].

A different approach is followed in the present paper, for the following reason. Our goal is to examine how kernel tricks can be used to alleviate the specific difficulties of relational learning; in relational terms, the question is about the quality of the propositionalization achieved through relational kernels. In other words, the focus is on the competence of the representation (the capacity of the hypothesis search space defined after the MI kernel) as opposed to, the competence of a particular algorithm (the average quality of the hypotheses learned by this algorithm in this search space).

Accordingly, while the proposed methodology is still based on the generation of artificial problems, it focuses on the kernel-based propositionalization of the MI examples. Formally, to each training set $\mathcal{L}$ is associated the propositional representation $\mathcal{R}_\mathcal{L}$, characterizing every MI example $\mathbf{x}$ as the ℓ-dimensional real-valued vector defined as $\Psi_\mathcal{L}(\mathbf{x}) = (K(\mathbf{x}_1, \mathbf{x}), \ldots, K(\mathbf{x}_\ell, \mathbf{x}))$.

By construction [20], any MI-SVM hypothesis h is expressed as a linear hypothesis in $\mathcal{R}_\mathcal{L}$, $h(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i.y_i.K(\mathbf{x}_i, \mathbf{x}) + \beta$, subject to ℓ inequality constraints:

$$\forall i = 1 \ldots \ell \qquad \alpha_i \geq 0 \tag{2}$$

Let $\mathcal{T}$ denote a t-example dataset propositionalized after $\mathcal{R}_\mathcal{L}$; the existence of a separating hyperplane for $\mathcal{T}$ is formalized as a set of t inequality constraints:

$$\forall j = 1 \ldots t \qquad y'_j.h(\mathbf{x'}_j) = y'_j.\left(\sum_{i=1}^{N} \alpha_i.y_i.K(\mathbf{x}_i, \mathbf{x'}_j) + \beta\right) \geq 1 \tag{3}$$

Let $\mathrm{Q}(\mathcal{L}, \mathcal{T})$ be defined as the set of inequality constraints (2) and (3). $\mathrm{Q}(\mathcal{L}, \mathcal{T})$ admits a solution iff the MI-SVM propositionalization defined from $\mathcal{L}$ has the capacity to separate the examples in $\mathcal{T}$. Note that the linear programming problem[1] (LPP) $\mathrm{Q}(\mathcal{L}, \mathcal{T})$ is much easier than the standard learning problem of whether the hypothesis actually learned from $\mathcal{L}$ will correctly classify $\mathcal{T}$. $\mathrm{Q}(\mathcal{L}, \mathcal{T})$ is an easier problem as it explicitly exploits the labels of the test examples (i.e., cheats) in order to find the $\ell + 1$ coefficients α_i and β; further, it can select *a posteriori* some of the SVM hyper-parameters, e.g. the error cost C.

The central argument of the paper is: $\mathrm{Q}(\mathcal{L}, \mathcal{T})$ gives deep insights into the quality of the propositionalization based on the kernel trick. Formally we show that the probability for $\mathrm{Q}(\mathcal{L}, \mathcal{T})$ to admit a solution, referred to as LPP satisfiability rate, induces a lower bound on the MI-SVM generalization error.

Proposition
Within a MI-SVM setting, let $\mathcal{L}$ be a training set of size ℓ, $\mathcal{R}_\mathcal{L}$ the associated kernel-based propositionalization, and $p_\mathcal{L}$ the generalization error of the optimal linear classifier $h^*_\mathcal{L}$ defined on $\mathcal{R}_\mathcal{L}$. Let $\mathbb{E}_\ell[p_\mathcal{L}]$ denote the expectation of $p_\mathcal{L}$ conditionally to $|\mathcal{L}| = \ell$.

Let a MI-SVM problem be defined as a pair of example sets $(\mathcal{L}, \mathcal{T})$. Considering a sequence of R independent MI-SVM problems $(\mathcal{L}_i, \mathcal{T}_i)$ such that the size of $\mathcal{L}_i$ (respectively $\mathcal{T}_i$) is ℓ (resp. t), let $\varepsilon_R(\ell, t)$ denote the fraction of LPPs $\mathrm{Q}(\mathcal{L}_i, \mathcal{T}_i)$ that are satisfiable. Then for any $\eta > 0$, with probability at least $1 - exp(-2\eta^2 R)$,

$$\mathbb{E}_\ell[p_\mathcal{L}] \geq 1 - (\varepsilon_R(\ell, t) + \eta)^{\frac{1}{t}}. \tag{4}$$

Proof
Given $\mathcal{L}$, $h^*_\mathcal{L}$ and $p_\mathcal{L}$ as above, the probability for a t example set $\mathcal{T}$ to include no example misclassified by $h^*_\mathcal{L}$ is $(1 - p_\mathcal{L})^t$.

[1] Actually, this problem should rather be viewed as a constraint satisfaction problem on continuous variables, as it does not involve any optimization objective; the only point is whether the set of linear inequalities admits a solution.

It is straightforward to see that if $\mathcal{T}$ does not contain examples that are misclassified by $h^*_{\mathcal{L}}$, $\mathrm{Q}(\mathcal{L},\mathcal{T})$ is satisfiable. Therefore the probability for $\mathrm{Q}(\mathcal{L},\mathcal{T})$ to be satisfiable conditionally to $\mathcal{L}$ is greater than $(1-p_{\mathcal{L}})^t$:

$$\mathbb{E}_{|\mathcal{T}|=t}[\ \mathrm{Q}(\mathcal{L},\mathcal{T})\ \text{satisfiable}] \geq (1-p_{\mathcal{L}})^t$$

Taking the expectation of the above w.r.t. $|\mathcal{L}|=\ell$, it comes:

$$\mathbb{E}_{|\mathcal{T}|=t,\ |\mathcal{L}|=\ell}[\ \mathrm{Q}(\mathcal{L},\mathcal{T})\ \text{satisfiable}] \geq \mathbb{E}_{|\mathcal{L}|=\ell}[(1-p_{\mathcal{L}})^t] \geq (1-\mathbb{E}_\ell[p_{\mathcal{L}}])^t \qquad (5)$$

where the right inequality follows from Jensen's inequality. Next step is to bound the left term from its empirical estimate $\varepsilon_R(\ell,t)$, using Hoeffding's bound. With probability at least $1-exp(-2\eta^2 R)$,

$$\mathbb{E}_{|\mathcal{T}|=t,\ |\mathcal{L}|=\ell}[\ \mathrm{Q}(\mathcal{L},\mathcal{T})\ \text{satisfiable}] < \varepsilon_R(\ell,t)+\eta \qquad (6)$$

From (5) and (6) it comes that with probability at least $1-exp(-2\eta^2 R)$

$$(1-\mathbb{E}_\ell[p_{\mathcal{L}}])^t \leq \varepsilon_R(\ell,t)+\eta$$

which concludes the proof. □

This theoretical result allows us to draw conclusions about the quality (generalization error) of the MI-SVM framework, based on the experimental satisfiability rate of the linear programming problem $\mathrm{Q}(\mathcal{L},\mathcal{T})$.

3.2 Order Parameters and Experimental Setting

The satisfiability of $\mathrm{Q}(\mathcal{L},\mathcal{T})$ is systematically investigated following the PT paradigm [3], based on the definition of order parameters. These order parameters, summarized in Table 1 together with their range of variation in the experiments, intend to characterize the key complexity factors in a MI-SVM problem, related to the instances, the examples, and the target concept.

Table 1. Order parameters for the MI LPP, and range of variation in the experiments

d	Dimension of the instance space $\mathcal{X}=[0,1]^d$	30
m	Number of sub-concepts in the target concept	30
ε	Coverage of a sub-concept $=\varepsilon^d$	.15
ℓ	Number of training examples	60 (30 +, 30 −)
t	Number of test examples	200 (100 +, 100 −)
N, N'	Number of instances in pos./neg. example	100
n	Number of relevant instances per positive example	30...100
n'	Number of relevant instances per negative example	0...100
nm	Number of sub-concepts not satisfied by neg. examples	10,20,25

Instance space $\mathcal{X}$ is set to $[0,1]^d$; unless specified otherwise, any instance x is uniformly drawn in $\mathcal{X}$. We denote $\mathcal{B}_\varepsilon(x)$ the ε-radius ball centered on x w.r.t. L_∞

norm. The target concept involves m sub-concepts C_i ; $C_i(x)$ holds iff x belongs to $B_\varepsilon(z_i)$, where z_i is a uniformly drawn instance. For $m > 1$ (resp. $m = 1$) such a target concept follows the *presence-based* (resp. *linear*) MI setting (section 2), Positive (respectively negative) examples include N (resp. N') instances. An instance is said to be *relevant* if it satisfies some sub-concept. An example is said to satisfy a sub-concept if it includes an instance satisfying this sub-concept. Positive (respectively negative) examples involve n (resp. n') relevant instances. Any negative example fails to satisfy exactly nm (for near-miss) sub-concepts. Naturally, $n \geq m$ and $nm \geq 1$.

For each order parameter setting, 40 pairs (training set $\mathcal{L}$, test set $\mathcal{T}$) are built, made of an equal number of positive and negative iid examples; each example involves the required number of relevant instances, uniformly drawn in some $B_\varepsilon(z_i)$, and other instances uniformly drawn in $\mathcal{X}$, conditionally to parameters N and n for positive examples (resp., N', n' and nm for negative examples). Set $\mathcal{T}$ is propositionalized after $\mathcal{R}_\mathcal{L}$, using Gaussian instance kernels with parameter $\sigma = 1$; the bag kernel uses the number of example instances as normalising function (eq. 1).

3.3 Goal of the Experiments

The paper goal is to see whether the MI-SVM framework overcomes the specific difficulties of relational learning, and whether a phase transition phenomenon occurs. The first goal of the experiments is to assess the satisfiability of the LPP; it is expected that the problem is satisfiable, i.e. positive and negative test examples can be discriminated, as far as their number of relevant instances are sufficiently different ($n <> n'$); the question thus is whether the diagonal region $n = n'$ is a critical region, and if it is the case, what its width is. This goal is achieved by measuring the LPP satisfiability, averaged over 40 problems $(\mathcal{L}_i, \mathcal{T}_i)$ independently generated for each order parameter setting.

The second goal is to assess the actual relation between the LPP satisfiability and the MI-SVM generalization error, in other words the relevance of the proposed approach. Indeed the lower bound on the MI-SVM generalization error based on the satisfiability does not say much as only $R = 40$ problems are considered per order parameter setting for computational feasibility. It thus remains to see whether the critical LPP region is also critical from a MI-SVM point of view, i.e. if it is a region where the standard test error is high too. This goal is classically achieved by learning a MI-SVM hypothesis from $\mathcal{L}_i$, measuring its error on $\mathcal{T}_i$, and averaging the test error over all problems generated for each order parameter setting.

4 Experiments

This section reports on the extensive experimental study conducted after the order parameters (Table 1). In total, 30,000 artificial MI-SVM problems have been considered. Let us first summarize the lessons learned before detailing and discussing the results.

4.1 Summary of the Results

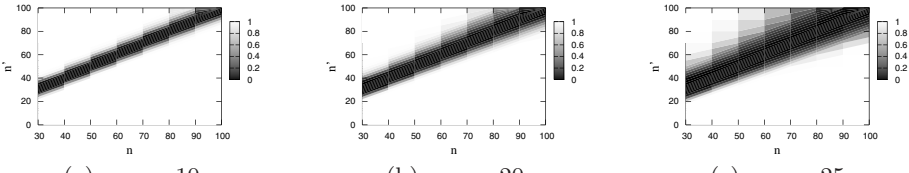
Firstly, the existence of an unsatisfiable region is experimentally demonstrated (Fig. 1). As expected, the unsatisfiable region corresponds to "truly relational" problems, e.g. when no distinction can be made between positive and negative examples based on their number of relevant instances ($n' = n$). Surprisingly, the width of the unsatisfiable region increases as parameter nm increases, i.e. when few sub-concepts are satisfied by a negative example. An interpretation for these findings is proposed in section 4.2.

Secondly, the unsatisfiable region is also a critical region from a MI-SVM learning viewpoint, which confirms the practical relevance of the lower bound established in section 3.1. The learning accuracy decreases smoothly but significantly while the satisfiability rate abruptly goes to 0 (Fig. 3); in the unsatisfiable region, the average test error is circa 40%.

4.2 LPP Satisfiability Landscape

Each LPP has been solved using the GGLPK package, with an average resolution cost of 16 seconds (on PC Pentium IV, 3.0 Ghz).

The average satisfiability computed for each order parameter setting mostly depends on the number n and n' of relevant instances in positive and negative examples. For the sake of readability, the satisfiability is thus graphically displayed in the (n, n') plane; the color of pixel (x, y) is black (respectively white) if all LPP with $(n = x, n' = y)$ are unsatisfiable (resp. satisfiable). Fig. 1 shows the unsatisfiable black region, centered on the diagonal $n = n'$.



(a) $nm = 10$ (b) $nm = 20$ (c) $nm = 25$

Fig. 1. LPP satisfiability versus n and n', averaged over 40 runs, for various values of the number nm of sub-concepts not satisfied by a negative example. All other order parameter values are as in Table 1.

These results are explained from the distribution of the examples in the kernel-based propositional space. Fig. 2 illustrates this distribution in a propositionalized plane where the two attributes are derived from a positive and a negative training example. Let the instance kernel be the Gaussian kernel[2]. Let $\bar{k}_C$ and $\bar{k}_U$ respectively denote the expectation of $k(x, x')$ for two instances satisfying the

[2] The interpretation only considers the Gaussian case; however complementary experiments done with polynomial kernels lead to similar LPP unsatisfiability landscape.

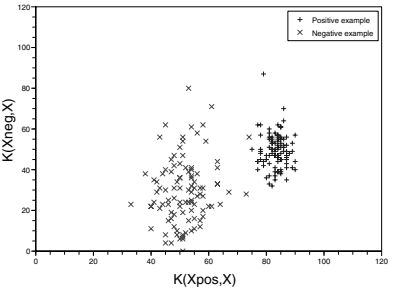


Fig. 2. Distribution of kernel-based propositionalized examples (legend + for positive, $\times$ for negative), with $n = 50$, $n' = 30$, $nm = 10$. First (second) coordinate corresponds to $K(\mathbf{x}, \cdot)$ with $\mathbf{x}$ a positive (negative) training example.

same sub-concept C (resp., uniformly drawn). Considering MI examples $(\mathbf{x}, y)$ and $(\mathbf{x}', y')$, the expectation of $K(\mathbf{x}, \mathbf{x}')$ is thus analytically derived:

$$\mathbb{E}[K(\mathbf{x}, \mathbf{x}')] = \begin{cases} \frac{1}{m}(\frac{n}{N})^2(\bar{k}_C - \bar{k}_U) + \bar{k}_U & \text{if } y = y' = 1 \\ \frac{1}{m}(\frac{n'}{N'})^2(\bar{k}_C - \bar{k}_U) + \bar{k}_U & \text{if } y = y' = -1 \\ \frac{1}{m}\frac{n}{N}\frac{n'}{N'}(\bar{k}_C - \bar{k}_U) + \bar{k}_C & \text{if } y \neq y' \end{cases}$$

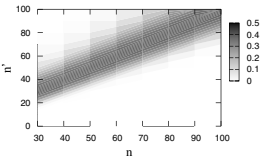
Therefore in the neighborhood of the diagonal region[3] $n = n'$, the distribution of the propositionalized examples hardly depends on their class, adversely affecting the discrimination task.

The fact that the width of the unsatisfiable region increases with the number nm of sub-concepts that are not satisfied by negative examples can be explained along the same lines. As nm increases, so does the variance of the distribution of the propositionalized negative examples, thus increasing the overlap between the distribution of positive and negative examples.

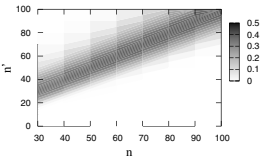
4.3 Generalization Error Landscape

As already mentioned, the lower bound given in section 3.1 is poorly informative with respect to the generalization error; an unsatisfiability rate of 100 % over 40 problems only allows us to conclude that the generalization error is greater than 0.8 % with confidence 95%. To estimate the tightness of the bound, the actual generalization error was thus estimated empirically by learning from the training set and measuring the error on the test set, averaged over all problems generated for each order parameter setting. Each MI-SVM problem was solved using SVMTorch [7] with an average computational cost of 25 seconds (on PC Pentium IV, 3.0 Ghz). For the sake of readability, the error is graphically displayed in the (n, n') plane; the color of pixel (x, y) depicts the average error for

[3] Actually, the unsatisfiable region corresponds to $\frac{n}{N} = \frac{n'}{N'}$. For simplicity, the distinction is omitted in the paper as $N = N'$.



(a) $C = 100$



(b) $C = 1{,}000{,}000$

Fig. 3. Generalization error of MI-SVM in the (n, n') plane, estimated from SVMTorch test error averaged on 40 problems, for cost error $C = 10^2$ and 10^6

$(n = x, n' = y)$; a white pixel stands for no error while a black pixel stands for 50% error (same as random guessing).

Indeed the SVMTorch parameters were not optimized for each problem. Still, experiments done with the cost error C ranging in $10, \ldots, 10^6$ lead to the same general picture, and confirm that the MI-SVM error increases with the LPP unsatisfiability (Fig. 3). While the unsatisfiability rate abruptly goes to 100%, the error rate increases more gently, but significantly; when the unsatisfiability is above 80% the average test error is above 30%.

5 Conclusion and Perspectives

The contribution of the paper is twofold. Firstly, a relaxed formalization of kernel-based learning in terms of linear programming has been defined, and it has been shown that the LPP satisfiability rate induces a lower bound on the generalization error. Contrasting with the mainstream asymptotic framework [20], the presented analysis is relevant for small size datasets, which makes sense indeed in application domains such as chemometry [16].

Secondly, the LPP framework has been used to demonstrate the existence of a phase transition phenomenon for standard MI-SVM kernels; further, the LPP unsatisfiable region corresponds to a critical region from a MI-SVM learning standpoint, where the test error is consistently greater than 30% after an extensive empirical study on artificial problems.

Further research will consider more sophisticated MI-SVM approaches [1,8], and see whether they also present a phase transition phenomenon in relation with the specific difficulties of *presence-based* MI learning. Another direction perspective is to further investigate the LPP framework, using the satisfiability rate as a criterion for kernel selection, or active learning.

Acknowledgments

The authors thank Olivier Teytaud for fruitful discussions, and gratefully acknowledge the support of the Network of Excellence PASCAL, IST-2002-506778.

References

1. Andrews, S., Tsochantaridis, I., Hofmann, T.: Support Vector Machines for Multiple-Instance Learning. In: NIPS Proc. of 15th, pp. 561–568 (2002)
2. Blockeel, H., Page, D., Srinivasan, A.: Multi-Instance Tree Learning. In: ICML, pp. 57–64 (2005)
3. 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)
4. Cheeseman, P., Kanefsky, B., Taylor, W.: Where the Really Hard Problems are. In: IJCAI, pp. 331–337 (1991)
5. Chen, Y., Wang, J.Z.: Image Categorization by Learning and Reasoning with Regions. Journal of Machine Learning Research 5, 913–939 (2004)
6. Chevaleyre, Y., Zucker, J.-D.: Solving Multiple-Instance and Multiple-Part Learning Problems with Decision Trees and Rule Sets. Application to the Mutagenesis Problem. In: Canadian Conference on Artificial Intelligence, pp. 204–214 (2001)
7. Collobert, R., Bengio, S., Mariéthoz, J.: Torch: A Modular Machine Learning Software Library. Technical Report IDIAP-RR 02-46 (2002)
8. Cuturi, M., Vert, J.-P.: Semigroup Kernels on Finite Sets. In: NIPS, pp. 329–336 (2004)
9. Dietterich, T., Lathrop, R., Lozano-Perez, T.: Solving the Multiple-Instance Problem with Axis-Parallel Rectangles. Artificial Intelligence 89(1-2), 31–71 (1997)
10. Gärtner, T., Flach, P.A., Kowalczyk, A., Smola, A.J.: Multi-Instance Kernels. In: ICML, pp. 179–186 (2002)
11. Giordana, A., Saitta, L.: Phase Transitions in Relational Learning. Machine Learning 41, 217–251 (2000)
12. Hogg, T., Huberman, B.A., C., Williams, C.P.: Phase Transitions and the Search Problem. Artificial intelligence 81(1-2), 1–15 (1996)
13. Kearns, M., Li, M.: Learning in the Presence of Malicious Errors. SIAM J. Comput. 22, 807–837 (1993)
14. Kramer, S., Lavrac, N., Flach, P.: Propositionalization Approaches to Relational Data Mining. In: Dzeroski, S., Lavrac, N. (eds.) Relational data mining, pp. 262–291 (2001)
15. Kwok, J., Cheung, P.-M.: Marginalized Multi-Instance Kernels. In: Kwok, J., Cheung, P.-M. (eds.) IJCAI, pp. 901–906 (2007)
16. Mahé, P., Ralaivola, L., Stoven, V., Vert, J.-P.: The Pharmacophore Kernel for Virtual Screening with Support Vector Machines. Journal of Chemical Information and Modeling 46, 2003–2014 (2006)
17. Maron, O., Lozano-Pérez, T.: A Framework for Multiple-Instance Learning. In: NIPS, pp. 570–576 (1997)
18. Muggleton, S., De Raedt, L.: Inductive Logic Programming: Theory and Methods. Journal of Logic Programming 19, 629–679 (1994)
19. Pernot, N., Cornuéjols, A., Sebag, M.: Phase Transitions Within Grammatical Inference. In: Pernot, N., Cornuéjols, A., Sebag, M. (eds.) IJCAI, pp. 811–816 (2005)
20. Vapnik, V.N.: Statistical Learning Theory. Wiley-Interscience, Chichester (1998)
21. Weidmann, N., Frank, E., Pfahringer, B.: A Two level Learning Method for Generalized Multi-Instance Problems. In: Lavrač, N., Gamberger, D., Todorovski, L., Blockeel, H. (eds.) ECML 2003. LNCS (LNAI), vol. 2837, pp. 468–479. Springer, Heidelberg (2003)
22. Zhang, Q., Goldman, S.A.: EM-DD: A Improved Multiple-Instance Learning Technique. In: NIPS Proc of the 14th, pp. 1073–1080 (2001)

Combining Clauses with Various Precisions and Recalls to Produce Accurate Probabilistic Estimates

Mark Goadrich and Jude Shavlik

University of Wisconsin - Madison

Abstract. Statistical Relational Learning (SRL) combines the benefits of probabilistic machine learning approaches with complex, structured domains from Inductive Logic Programming (ILP). We propose a new SRL algorithm, GleanerSRL, to generate the probability that an example is positive within highly-skewed relational domains. In this work, we combine clauses from Gleaner, an ILP algorithm for learning a wide variety of first-order clauses, with the propositional learning technique of support vector machines to learn well-calibrated probabilities. We find that our results are comparable to SRL algorithms SAYU and SAYU-VISTA on a well-known relational testbed.

1 Introduction

Inductive Logic Programming (ILP) is the process of learning first-order clauses to correctly categorize domains of relational data. ILP uses relations expressed in mathematical logic to describe examples, and can handle variable-sized structures and sequences [5]. Statistical Relational Learning (SRL) [8] builds on the benefits of relational data and introduces methods for learning from large and noisy datasets, typically in combination with producing probabilistic outputs as opposed to strict classifications. Prominent work within SRL includes the generative approaches of Probabilistic Relational Models by Friedman et al. [7] and Markov Logic Networks from Richardson and Domingos [17], as well as discriminative algorithms such as SAYU and SAYU-VISTA from Davis et al. [4].

In this work we propose the use of Gleaner [9] as the foundation for a new discriminative SRL algorithm called GleanerSRL. Gleaner is a two-stage algorithm developed to first learn a broad spectrum of clauses and then combine them into thresholded theories aimed at maximizing precision for a particular choice of recall. Gleaner can run quickly on large datasets when one has a set of available processors. Already new desktop computers include multiple cpu's (called 'cores'), and within a few years it will be common for desktop computers to have 32, 64, 128, or more cores. Also we have previously shown that Gleaner can achieve good performance from only a relatively small number of clause evaluations per seed, because it keeps more than one good clause per seed, and we believe the clauses learned from Gleaner will be more diverse than those found with other approaches.

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 122–131, 2008.

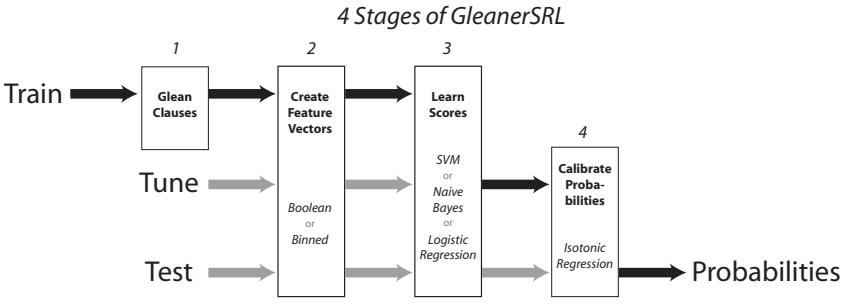


Fig. 1. GleanerSRL takes training, tuning and testing examples and returns a probability estimate for the testing examples after four stages of processing. Black arrows denote dependencies in training for a stage, while grey arrows denote only data transformations. Note that the testset is not examined until training is complete in order to allow us unbiased estimates of future performance.

We modify the two-stage approach used by Gleaner into GleanerSRL, which learns clauses, produces feature vectors, and generates probabilities. We then evaluate the quality of these approaches using Mean Cross Entropy in comparison to SAYU and SAYU-VISTA. Finally, we conclude by discussing future directions and related work.

2 Learning Probabilities with GleanerSRL

GleanerSRL is a four-stage algorithm to directly estimate probabilities for relational domains, as shown in Figure 1. The first stage learns a wide variety of clauses from a large number of seed examples. The second stage uses the clauses learned to generate a feature vector for each example, while the third stage uses this feature vector in propositional learners to learn a numeric score for each example, and the fourth stage calibrates these scores into probabilities. In essence, we will be transforming our tasks into propositional domains through the medium of our learned clauses and then using standard propositional learners to estimate these probabilities.

2.1 Gleaning Clauses

The first stage of GleanerSRL is identical to that of the original Gleaner, and learns a wide spectrum of clauses, illustrated in Figure 2. Gleaner brings in a training set of positive and negative examples along with the background knowledge. Each clause examined will cover a subset of examples; those that are positive we call true positives (TP) and those that are negative we call false positives (FP). The precision of a clause is then defined as $\frac{TP}{TP+FP}$. Not all positives will be necessarily covered by a clause; those that are missed are called false negatives (FN), and we define the recall of a clause as $\frac{TP}{TP+FN}$.

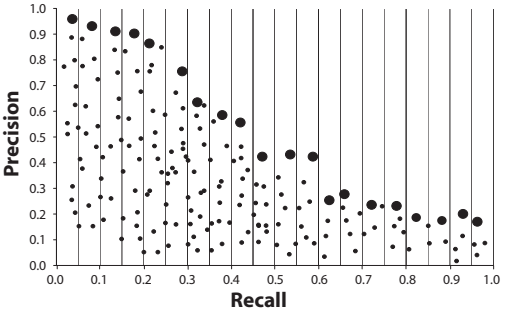


Fig. 2. A hypothetical run of Gleaner for one seed and 20 bins on the training set, showing each considered clause as a small circle, and the chosen clause per bin as a large circle. This is repeated for K seeds to gather $B \times K$ clauses (assuming a clause is found that falls into each bin for each seed).

Gleaner uses Aleph [18] to search for clauses using K seed examples to encourage diversity. The recall dimension is uniformly divided into B equal sized bins; in our experiments that appear in Section 3 our bins will be $[0, 0.05], [0.05, 0.10], \ldots, [0.95, 1]$. For each seed, we consider up to N possible clauses using stochastic local-search methods [10]. As these clauses are generated, we compute the recall of each clause and determine into which bin the clause falls. Each bin keeps track of the best clause appearing in its bin for the current seed. We use the heuristic function *precision* $\times$ *recall* to determine each bin's best clause, since we believe this will increase the generality of our clauses.

At the end of this search process, there will be B clauses collected for each seed and K seed examples for a maximum of $B \times K$ clauses (assuming a clause is found that falls into each bin for each seed). Since clauses can be learned independently for each seed, Gleaner is fast for large datasets because each seed can be explored in parallel.

2.2 Creating Features

Whereas the second stage of Gleaner combines these learned clauses in an attempt to maximize precision-recall (PR) curves on an unseen testset, here we wish to instead estimate the probability that an example is positive. We cannot directly convert Gleaner's final PR curve into numeric scores, since each point in the test curve may come from a distinct theory and threshold combination. Gleaner requires the user to find the point closest to their desired recall and then uses this theory to rank the testset examples based on the particular theory which generated this point. Since we are interested in directly generating probabilities instead of recall-precision curves, we introduce here a new second stage for GleanerSRL to transform the learned clauses into propositional features.

Our first transformation is the *Boolean* feature method. We create one feature for each clause and assign the feature a value of 1 if the clause is true and 0 if the

clause is false. In a scenario with 20 bins and 100 seeds, this would generate 2000 features, given that there is a clause found within each bin for all seeds and all clauses are unique. We have found in practice that there are many less features generated than the complete 2000 due to duplicate clauses within the high-recall bins. These Boolean feature vectors are created for the trainset, tuneset and testset examples.

A second approach is the *binned* feature method. We make use of the theories and thresholds as previously calculated by the second stage Gleaner, making one feature per bin. For each example, the value of a feature is equal to the cumulative precision of each clause in this bin's theory that match this example. This reduces our features to only the number of bins no matter how many seeds are explored. In our earlier work with Gleaner, we noticed that duplicate clauses were found more often in the high-recall bins. This binning feature method will retain a more uniform coverage of the recall space and will also take advantage of combining similar clauses. We look at two binning feature methods, one with the features as raw score of the cumulative precision for each bin, and one with the cumulative precision *normalized* to between 0 and 1 by the maximum score found in that bin. The precision for each clause is calculated on the trainset, and bin feature vectors are created for the trainset, tuneset and testset examples. Using the tuneset to calculate the precision is also recommended, but I reuse the trainset to maintain a suitably large number of positive examples for these calculations.

2.3 Learning to Predict Scores

With the feature vectors calculated from the second stage, our problem is now propositional in nature. The third stage of GleanerSRL uses standard propositional approaches to estimate the probability for each example. We will be using classifiers where each feature f is assigned a weight w_f through training. For a new example x_i, where $0 < i \leq N$ for a testing set of size N and $x_{i,j}$ is the value of feature f on example x_i, we discriminate between positive and negative using a threshold b as follows:

$$\text{If} \sum_{f \in feats} (w_f \times x_{i,f}) > b \quad \text{then +, else -}$$

We can achieve a richer feature space by using a kernel matrix K to give us a notion of similarity between example x_i and the examples in our training set (minus those set aside in the tuning set). The simplest kernel is constructed by taking the dot product of x_i and example x_j, such that $K(x_i, x_j) = \sum_{f \in feats}(x_{i,f} \times x_{j,f})$. We can then replace our weighted feature model from above with

$$\text{If} \sum_{j \in examples} (\alpha_j \times K(x_i, x_j)) > b \quad \text{then +, else -}$$

where α_j is a weight on each kernel-induced feature. For the purposes of estimating probabilities, we are only really interested in the weighted sum from the

Table 1. We examine five different kernel methods for calculating $K(x_i, x_j) = \sum_{f \in features} k(x_{i,f}, x_{j,f})$ for Boolean feature vectors

Kernel	$k(x_{i,f}, x_{j,f})$
Dot-Prod	$k(1,1) \ : \ 1, \ \text{else} : 0$
Precision	$k(1,1) \ : \ precision_f, \ \text{else} : 0$
Recall	$k(1,1) \ : \ 1 - recall_f, \ \text{else} : 0$
Both-Pos	$k(1,1) \ : \ precision_f^2, \ \text{else} : 0$
Info	$k(1,1) \ : \ -log_2((\frac{TP+FP}{\lvert trainset \rvert})^2)$ $k(1,0)$ or $k(0,1) \ : \ -log_2(2 \times \frac{TP+FP}{\lvert trainset \rvert} \times (1 - \frac{TP+FP}{\lvert trainset \rvert}))$ $k(0,0) \ : \ -log_2((1 - \frac{TP+FP}{\lvert trainset \rvert})^2)$

above thresholded classification, and we use this as a numeric score s for each example.

Our particular classifier choice for this paper is the Support Vector Machine (SVM) [2]. SVMs learn weights for α_j that maximize the margin between the classification hyperplane and the training data by solving a linear or quadratic program. In practice (especially when using linear programming), most α_j values will be 0, thus ignoring a large number of our kernel-induced features. In our preliminary testing, we also investigated using naïve Bayes and Logistic Regression. We found them to be significantly outperformed by the SVM approach and therefore do not include those results.

We examine here five different kernels, shown in Table 1, for use within our SVM. First we use a simple *dot-product* kernel discussed above in combination with both of the binning feature methods. As for kernels on Boolean features, we also use a dot-product kernel, as well as four attempts to incorporate statistics from the training set about each clause.

Since the similarity under the dot-product kernel is only increased when two examples match on a feature (when features are all Boolean valued), we can score each match instead by the *precision* of that clause as calculated on the training set example. This means that examples will be more similar when they are both covered by high-precision clauses. Similarly for *recall*, we use the score $1 - recall_f$. Since we aim to collect clauses that have high precision in the first stage, matching an example on a low-recall clause should be more meaningful in relation to the positive examples.

We also explore two kernel methods related to the probability that a given clause is expected to match a particular example, called *both-pos* and *info*. Precision equals the probability of an example being truly positive given that it matched the clause, therefore $precision_f^2$ is the probability that any two examples are truly positive given that they both match on $x_{i,f}$, assuming independence, and we use this as our weight for *both-pos*. The actual probability of a given clause matching any example is based on the number of true and false

positives for that clause: $prob_f = \frac{TP+FP}{|dataset|}$. For the *info* kernel, we consider the information content for the probabilities of both, only one, or none of the examples matching (using $-log_2(p(X))$ for each case). Two other kernel method explored but not reported here are the Hamming distance between two clauses (where clauses are more similar if they return the same classification on a given example, be it positive or negative) and a Gaussian kernel, as they were outperformed by our above kernels in preliminary tests.

2.4 Calibrating Probabilities

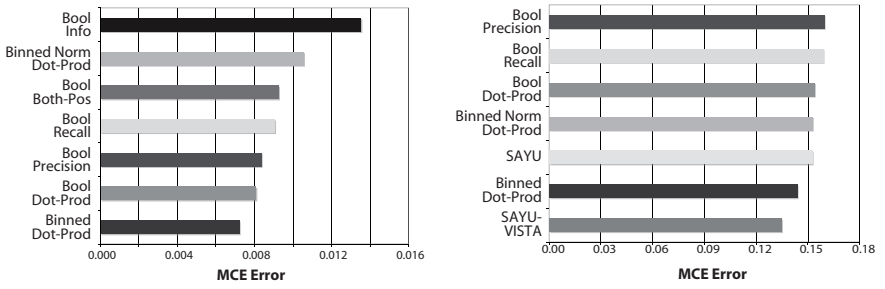
The SVM weighted sums from above will not be strict probabilities between 0 and 1. Therefore in the final stage of GleanerSRL, we calibrate these scores into proper probabilities. Zadrozny and Elkan [21] and Niculescu-Mizil and Caruna [14] recommend Isotonic Regression for large highly-skewed datasets. The main idea behind isotonic regression is to transform the sorted list of SVM scores into monotonically increasing probability scores which minimize the probability errors, and can be seen as an adaptive method for automatically finding the proper bin widths based on the tuning data. We achieve this isotonic regression by using the Pool Adjacent Violators (PAV) algorithm. Given a set of examples (s_i, c_i), where each example i consists of the weighted SVM score s along with the classification c, where c is now 1 for positive examples and 0 for negative examples, PAV will return a mapping for a range of s_i scores to their calibrated c_i values. We calibrate our probabilities on a tuning set and then use the found mapping to assign probabilities $p(x_i)$ on our testing set. Note that this step is not necessary but still recommended when using naïve Bayes or logistic regression.

3 Experimental Results

We follow the methodology of Caruna and Niculescu-Mizil [1], and evaluate our probability estimates from GleanerSRL using the metric of Mean Cross Entropy (MCE). In preliminary work we found the results from using mean squared error to be very similar, thus those results are not included here. Cross entropy calculates the difference of predicted probability from the true probability; the formula is derived from information theory and Kullback-Liebler divergence. Formally,

$$MCE = -\frac{\sum_{i=0}^{n}(a(x_i)log(p(x_i)) + (1 - a(x_i))log(1 - p(x_i))}{n}$$

where n is the testset size, $a(x_i)$ is the actual probability of example i (in our case 0 or 1) and $p(x_i)$ is our estimate. To properly compute these numbers, we enforced a bound on the probability estimates so that $0 < prob_{min} \leq p(X) \leq 1 - prob_{min} < 1$. We tune this bound on our tuning set using leave-one-out cross-validation. This bound is helpful when there is a complete mistake in probability, where the actual probability is 1 and the predicted probability is 0, or vice versa, since the cross entropy error will be infinity.



(a) Cross Entropy Error for OMIM (b) Cross Entropy Error for Advisor

Fig. 3. Comparison of Mean Cross Entropy GleanerSRL kernel methods and SAYU, ordered from least to most on each dataset

We report results on two highly-skewed domains, OMIM and Advisor:

OMIM. This is the Online Mendelian Inheritance in Man genetic-disorder biomedical information extraction dataset from Ray and Craven [16]. From a sentence such as "Mutations in the COL3A1 gene have been implicated as a cause of type IV Ehlers-Danlos syndrome, a disease leading to aortic rupture in early adult life," the task is to extract a relationship between the gene COL3A1 and Ehlers-Danlos syndrome. We use the ILP dataset construction of Goadrich et al. [9], which contains five disjoint folds with 233 positive and 103,959 negative examples.

Advisor. This dataset is derived from the University of Washington CS Department. It was constructed by Richardson and Domingos [17]. The goal is to predict the advisor of a graduate student, where students, professors, courses and papers are known to be related by author, instructor, and teaching assistant relations. This dataset contains five disjoint folds with a total of 113 positive examples and 2,711 negative examples.

For the parameters of GleanerSRL, we ran Gleaner with 20 bins, 100 seeds for OMIM and 50 seeds for Advisor until 25,000 clauses were examined for each seed. In combination with the SVM for stage three, we tuned with nine values for the complexity parameter C ranging from 10,000 to 0.0001, and in stage four we tested nine values for $prob_{min}$ from 0.25 to 0.0001. Different C and $prob_{min}$ values were chosen for each fold.

Figure 3(a) shows the results of our kernel choices for GleanerSRL on OMIM. Binnned feature vectors combined with the dot product kernel outperforms the rest, however, this is only a statistically significant difference with the Boolean match kernel. It is interesting to note that the highest scoring approaches use the dot product kernels for both types of feature vectors.

The results on Advisor in Figure 3(b) again show that Binned Dot Product outperforms our other approaches. Once again the dot product kernel is the best choice. We also compare to SAYU and SAYU-VISTA from Davis et al. [4], using a tree-augmented network [6] and an eager rule-adoption policy. SAYU learns a Bayesian network for classification by continually adding features when

a clause makes a significant improvement in the Area Under the Curve for Precision and Recall (AUC-PR). VISTA builds on SAYU by incorporating new predicates throughout the learning process. We find that the difference between GleanerSRL both SAYU-VISTA and SAYU is not statistically significant. We separately explored directly optimizing the MCE for SAYU, and found the results were slightly worse than optimizing for AUC-PR, but the difference was not statistically significant.

4 Related and Future Work

One typical approach to weighting a theory in ILP is propositionalization, where each clause in a theory is translated into a Boolean feature. This allows for a number of propositional learning algorithms to be used for learning weights on each clause. Pompe and Kononenko [15] use a naïve Bayes classifier to find their weights, while Srinivasan and King [19] use logistic regression, a technique to find weights that will maximize the likelihood of the data.

Koller and Pfeffer [11] learn the weights for clauses in a theory by first creating a Bayesian network model for the theory. They then use an Expectation Maximization algorithm to set the parameters to maximize the likelihood of the data. Their results are on a toy dataset with three clauses, so it is unknown how well this would extend to the very large datasets we propose to investigate here. Richardson and Domingos [17] extend work with Relational Markov Networks [20] to formulate Markov Logic Networks. Their setup can take clauses from either ILP or a domain expert, translate them to a Markov Network, and then learn the weights on the clauses using logistic regression. Davis *et al.* [3] compare naïve Bayes, TAN and the sparse candidate algorithm as alternate methods of learning appropriate weight parameters. As in the above methods, there is no attempt to modify the learned theory, only the weights.

Support Vector Machines are a recent addition to the SRL toolkit, with contributions of Support Vector Inductive Logic Programming (SVILP) from Muggleton et al. [13], and kFOIL by Landwehr et al. [12]. SVILP is most similar to our work, in that both use learned first-order clauses to create a kernel for probabilistic output. However, where they use mainly a Gaussian kernel with a prior probability over the clauses, we explore kernel methods and clause generation that are informed by precision and recall on the training set. kFOIL presents a dynamic kernel construction process, where the choice of clauses to add is informed by the current classification accuracy. Conversely, GleanerSRL learns clauses first and then constructs the kernel, and through the use of Gleaner we can quickly and in parallel explore a large area of large hypothesis spaces.

We have explored the use of GleanerSRL through comparisons on two relational domains. In future work, we plan to compare with other SRL methods and apply GleanerSRL to much larger testbeds, where we hope to see significant speedups in search time due to using Gleaner over other methods. We also plan to investigate other kernel methods and propositional learning algorithms, as well as alternate feature vector transformations.

Acknowledgements

We would like to thank Ameet Soni, Jesse Davis, Louis Oliphant, the UW Condor Group, and our anonymous reviewers for their helpful comments and suggestions regarding this work. This project is supported by DARPA grant HR0011-04-1-0007 and DARPA IPTO under contract FA8650-06-C-7606.

References

1. Caruana, R., Niculescu-Mizil, A.: An Empirical Comparison of Supervised Learning Algorithms. In: Proceedings of the 23rd International Conference on Machine Learning (2006)
2. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press, Cambridge (2000)
3. Davis, J., Dutra, I.C., Page, D., Costa, V.S.: Establish Entity Equivalence in Multi-Relation Domains. In: Proceedings of the International Conference on Intelligence Analysis, Vienna, Va (2005)
4. Davis, J., Ong, I., Struyf, J., Burnside, E., Page, D., Costa, V.S.: Change of Representation for Statistical Relational Learning. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (2007)
5. Džeroski, S., Lavrac, N.: An Introduction to Inductive Logic Programming. In: Relational Data Mining, pp. 48–66. Springer, Heidelberg (2001)
6. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian Network Classifiers. Machine Learning 29(2-3), 131–163 (1997)
7. Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning Probabilistic Relational Models. In: Proceedings of the 16th International Conference on Artificial Intelligence, pp. 1300–1309 (1999)
8. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning. MIT Press, Cambridge (2007)
9. Goadrich, M., Oliphant, L., Shavlik, J.: Gleaner: Creating Ensembles of First-order Clauses to Improve Recall-Precision Curves. Machine Learning 64, 231–262 (2006)
10. Hoos, H., Stutzle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann, San Francisco (2004)
11. Koller, D., Pfeffer, A.: Learning Probabilities for Noisy First-Order Rules. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI), Nagoya, Japan (August 1997)
12. Landwehr, N., Passerini, A., Raedt, L.D., Frasconi, P.: kFOIL: Learning Simple Relational Kernels. In: Proceedings of the 21st National Conference on Artificial Intelligence (2006)
13. Muggleton, S., Amini, A., Lodhi, H., Sternberg, M.: Support Vector Inductive Logic Programming. In: Proceedings of the 8th International Conference on Discovery Science (2005)
14. Niculescu-Mizil, A., Caruana, R.: Predicting Good Probabilities with Supervised Learning. In: Proceedings of the 22nd International Conference on Machine Learning, pp. 625–632 (2005)
15. Pompe, U., Kononenko, I.: Naive Bayesian Classifier within ILP-R. In: Fifth International Workshop on Inductive Logic Programming, pp. 417–436 (1995)

16. Ray, S., Craven, M.: Representing Sentence Structure in Hidden Markov Models for Information Extraction. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence (2001)
17. Richardson, M., Domingos, P.: Markov Logic Networks. Machine Learning 62, 107–136 (2006)
18. Srinivasan, A.: The Aleph Manual Version 4 (2003), http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/
19. Srinivasan, A., King, R.: Feature Construction with Inductive Logic Programming: A Study of Quantitative Predictions of Biological Activity Aided by Structural Attributes. In: Muggleton, S. (ed.) Proceedings of the 6th International Workshop on Inductive Logic Programming. Stockholm University, Royal Institute of Technology, pp. 352–367 (1996)
20. Taskar, B., Abbeel, P., Wong, M.-F., Koller, D.: Label and Link Prediction in Relational Data. In: IJCAI Workshop on Learning Statistical Models from Relational Data (2003)
21. Zadrozny, B., Elkan, C.: Transforming Classifier Scores into Accurate Multiclass Probability Estimates. In: The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2002)

Applying Inductive Logic Programming to Process Mining

Evelina Lamma, Paola Mello, Fabrizio Riguzzi, and Sergio Storari

ENDIF – Università di Ferrara – Via Saragat, 1 – 44100 Ferrara, Italy
{evelina.lamma,fabrizio.riguzzi,sergio.storari}@unife.it
DEIS – Università di Bologna – Viale Risorgimento, 2 – 40136 Bologna, Italy
pmello@deis.unibo.it

Abstract. The management of business processes has recently received a lot of attention. One of the most interesting problems is the description of a process model in a language that allows the checking of the compliance of a process execution (or trace) to the model. In this paper we propose a language for the representation of process models that is inspired to the SCIFF language and is an extension of clausal logic. A process model is represented in the language as a set of integrity constraints that allow conjunctive formulas as disjuncts in the head. We present an approach for inducing these models from data: we define a subsumption relation for the integrity constraints, we define a refinement operator and we adapt the algorithm ICL to the problem of learning such formulas. The system has been applied to the problem of inducing the model of a sealed bid auction and of the NetBill protocol. The data used for learning and testing were randomly generated from correct models of the processes.

Keywords: Process Mining, Learning from Interpretations, Business Processes, Interaction Protocols.

1 Introduction

Every organization performs a number of *business processes* in order to achieve its mission. Complex organizations are characterized by complex processes, involving many people, activities and resources. The performances of an organization depend on how accurately and efficiently it enacts its business processes. Formal ways of representing business processes have been studied in the area of business processes management (see e.g. [1]), so that the actual enactment of a process can be checked for compliance with a model.

Recently, the problem of automatically inferring such a model from data has been studied by many authors (see e.g. [2,3,4]). This problem has been called Process Mining or Workflow Mining. The data in this case consists of execution traces (or histories) of the business process. The collection of such data is made possible by the facility offered by many information systems of logging the activities performed by users.

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 132–146, 2008.

In this paper, we propose a novel representation language for describing process models and an approach to Process Mining that uses learning from interpretations techniques from ILP. The language is inspired to the $\mathcal{S}$CIFF one [5] and extends clausal logic by allowing more complex formulas as disjuncts in the head of clauses.

We show how an execution trace can be represented as an interpretation and how we can use our language to check its compliance with the model. Thus we can cast a process mining problem as a learning from interpretations problem. In particular, we considered the discriminant problem that is solved by ICL [6], where we have positive and negative interpretations and we want to find a clausal theory that discriminates the two. In our case we assume that we have compliant and non compliant traces of execution of a process and we want to find a theory that accurately classifies a new trace as compliant or non compliant.

We differ from traditional process mining research in three respects: first we perform mining from both compliant and non compliant traces, while traditionally only compliant traces are considered; second we learn a declarative representation of a process model, while usually more procedural representation have been induced, such as Petri nets, and third, we are able to consider structured atomic activities, thanks to the first order representation.

The fact of having positive and negative traces is not commonly considered in the literature on process mining but it is interesting in a variety of cases: for example, a bank may divide its transactions into fraudulent and normal ones and may desire to learn a model that is able to discriminate the two. In general, an organization may have two or more sets of process executions and may want to understand in what sense they differ.

The use of a declarative language for process models is advocated by other authors as well [7]. The use of a structured representation allows the system to take into account many different properties of activities that would otherwise be overlooked.

The paper is organized as follows. Section 2 discusses how we represent execution traces with logic programming and describes the language used to represent process models. Section 3 presents the learning technique we have adopted for performing Process Mining. Section 4 reports on the experiments performed. Section 5 discusses related works and finally Section 6 concludes the paper.

2 A Representation for Process Traces and Models

A *process trace* t is a sequence of events that are activity executions. Each event is described by a number of attributes. The only requirement is that one of the attributes describes the activity type. Other attributes may be the executor of the event or event specific information.

An example of a trace is

$\langle a, b, c \rangle$

that means that an activity of type a was performed first, then an activity of type b and finally an activity of type c.

A *process model* PM is a formula in a language. An interpreter of the language must exists that, when applied to a model PM and a trace t, returns answer yes if the trace is compliant with the description and false otherwise. In the first case we write $t \models PM$, in the second case $t \not\models PM$.

A bag of process traces L is called a *log*. Usually, in Process Mining, only compliant traces are used as input of the learning algorithm, see e.g. [2,3,4]. We consider instead the case where we are given both compliant and non compliant traces.

A process trace can be represented as an interpretation: each event is modeled with an atom whose predicate is the activity type and whose arguments store the attributes of the activity execution. Moreover, an extra argument is added to the atom indicating the position in the sequence. For example, the trace:

$\langle a, b, c \rangle$

can be represented with the interpretation

$\{a(1), b(2), c(3)\}$.

If the execution time is an attribute of the event, then the position in the sequence can be omitted.

Besides the trace, we may have some general knowledge that is valid for all traces. This information will be called *background knowledge* and we assume that it can be represented as a normal logic program B. The rules of B allow to complete the information present in a trace t: rather than simply t, we now consider $M(B \cup t)$, the model of the program $B \cup t$ according to Clark's completion [8].

The process language we consider is a subset of the $\mathcal{S}$CIFF language, originally defined in [9,5], for specifying and verifying interaction in open agent societies.

A process model in our language is a set of Integrity Constraints (ICs). An IC, C, is a logical formula of the form

$$Body \rightarrow \exists(ConjP_1) \vee \ldots \vee \exists(ConjP_n) \vee \forall\neg(ConjN_1) \vee \ldots \vee \forall\neg(ConjN_m) \quad (1)$$

where $Body$, $ConjP_i$ $i = 1, \ldots, n$ and $ConjN_j$ $j = 1, \ldots, m$ are conjunctions of literals built over event atoms, over predicates defined in the background or over built-in predicates. The quantifiers in the head apply to all the variables not appearing in the body. The variables of the body are implicitly universally quantified with scope the entire formula.

We will use $Body(C)$ to indicate $Body$ and $Head(C)$ to indicate the formula $\exists(ConjP_1) \vee \ldots \vee \exists(ConjP_n) \vee \forall\neg(ConjN_1) \vee \ldots \vee \forall\neg(ConjN_m)$ and call them respectively the *body* and the *head* of C. We will use $HeadSetP(C)$ to indicate the set $\{ConjP_1, \ldots, ConjP_n\}$ and $HeadSetN(C)$ to indicate the set $\{ConjN_1, \ldots, ConjN_m\}$.

$Body(C)$, $ConjP_i$ $i = 1, \ldots, n$ and $ConjN_j$ $j = 1, \ldots, m$ will be sometimes interpreted as sets of literals, the intended meaning will be clear from the context. We will call P *conjunction* each $ConjP_i$ for $i = 1, \ldots, n$ and N *conjunction* each $ConjN_j$ for $j = 1, \ldots, m$. We will call P *disjunct* each $\exists(ConjP_i)$ for $i = 1, \ldots, n$ and N *disjunct* each $\forall\neg(ConjN_j)$ for $j = 1, \ldots, m$.

An example of an IC is

$$\begin{aligned} & a(bob, T), T < 10 \\ \rightarrow & \exists T1(b(alice, T1), T < T1) \\ & \vee \\ & \forall T1 \neg (c(mary, T1), T < T1, T1 < T + 10) \end{aligned} \tag{2}$$

The meaning of the IC (2) is the following: if *bob* has executed action a at a time $T < 10$, then *alice* must execute action b at a time $T1$ later than T or *mary* must not execute action c for 9 time units after T.

An IC C is true in an interpretation $M(B \cup t)$, written $M(B \cup t) \models C$, if, for every substitution θ for which $Body$ is true in $M(B \cup t)$, there exists a disjunct $\exists(ConjP_i)$ or $\forall\neg(ConjN_j)$ that is true in $M(B \cup t)$. If $M(B \cup t) \models C$ we say that the trace t is *compliant* with C.

Similarly to what has been observed in [10] for disjunctive clauses, the truth of an IC in an interpretation $M(B \cup t)$ can be tested by running the query:

$? - Body, not(ConjP_1), \ldots not(ConjP_n), ConjN_1, \ldots, ConjN_m$

in a database containing the clauses of B and the atoms of t as facts.

If the N conjunctions in the head share some variables, then the following query must be issued

$? - Body, not(ConjP_1), \ldots not(ConjP_n),$

$not(not(ConjN_1)), \ldots, not(not(ConjN_m))$

that ensures that the N conjunctions are tested separately without instantiating the variables.

If the query finitely fails, the IC is true in the interpretation. If the query succeeds, the IC is false in the interpretation. Otherwise nothing can be said. It is the user's responsibility to write the background B in such a way that no query generates an infinite loop. For example, if B is acyclic then we will have termination for a large class of queries [11].

A process model H is true in an interpretation $M(B \cup t)$ if every IC is true in it and we write $M(B \cup t) \models H$. We also say that trace t is compliant with H.

The ICs we consider are more expressive than logical clauses, as can be seen from the query used to test them: for ICs, we have the negation of conjunctions, while for clauses we have only the negation of atoms. This added expressiveness is necessary for dealing with processes because it allows us to represent relations between the execution times of two or more activities.

3 Learning ICs Theories

In order to learn a theory that describes a process, we must search the space of ICs. To this purpose, we need to define a generality order in such a space.

IC C is *more general* than IC D if C is true in a superset of the traces where D is true. If $D \models C$, then C is more general than D.

Definition 1 (Subsumption). *An IC D* subsumes *an IC C, written $D \geq C$, iff it exists a substitution θ for the variables in the body of D or in the N conjunctions of D such that*

- $Body(D)\theta \subseteq Body(C)$ *and*
- $\forall ConjP(D) \in HeadSetP(D),\ \exists ConjP(C) \in HeadSetP(C) : ConjP(C) \subseteq ConjP(D)\theta$ *and*
- $\forall ConjN(D) \in HeadSetN(D),\ \exists ConjN(C) \in HeadSetN(C) : ConjN(D)\theta \subseteq ConjN(C)$

Theorem 1. $D \geq C \Rightarrow D \models C$.

Proof. We must prove that all the models of D are also models of C. Let θ be the substitution with which D subsumes C. Consider a model i of D. If $\nexists\delta$ such that $Body(C)\delta$ is true in i, then C is true in i.

If $\exists\delta$ such that $Body(C)\delta$ is true in i, then $Body(D)\theta\delta$ will be true in i because $Body(D)\theta\delta \subseteq Body(C)\delta$. So there must be a disjunct of $Head(D)\theta\delta$ that is true in i.

Suppose that the disjunct $\exists(ConjP(D))$ of D is such that $\exists(ConjP(D)\theta\delta)$ is true in i: C will contain a disjunct $\exists ConjP(C)$ such that $ConjP(C) \subseteq ConjP(D)\theta$, thus $ConjP(C)\delta \subseteq ConjP(D)\theta\delta$ and it holds that $\exists(ConjP(C)\delta)$.

Suppose that the disjunct $\forall\neg(ConjN(D))$ of D is such that $\forall\neg(ConjN(D)\theta\delta)$ is true in i: C will contain a disjunct $\forall\neg(ConjN(C))$ such that $ConjN(D)\theta \subseteq ConjP(C)$, thus $ConjN(D)\delta\theta \subseteq ConjP(C)\delta$ and it holds that $\forall\neg(ConjP(C)\delta)$.

Thus i is also a model of C

In order to define a refinement operator, we must first define the language bias. We use a language bias that consists of a set of IC templates. Each template specifies:

- a set of literals BS allowed in the body;
- a set of disjuncts HS allowed in the head. For each disjunct, the template specifies:
  - whether it is a P or an N disjunct,
  - the set of literals allowed in the disjunct.

Thus we can define a refinement operator in the following way: given an IC D, the set of refinements $\rho(D)$ of D is obtained by performing one of the following operations:

- adding a literal from the IC template for D to the body;
- adding a disjunct from the IC template for D to the head: the disjunct can be
  - a formula $\exists(d_1 \wedge \ldots \wedge d_k)$ where $\{d_1, \ldots, d_k\}$ is the set of literals allowed by the IC template for D for a P disjunct,
  - a formula $\forall\neg(d)$ where d is a literal allowed by the IC template for D for an N disjunct;
- removing a literal from a P disjunct in the head;
- adding a literal to an N disjunct in the head. The literal must be allowed by the language bias.

The learning problem we consider is an adaptation to ICs of the learning from interpretation setting of ILP:

Given

- a space of possible process models $\mathcal{H}$
- a set I^+ of positive traces;
- a set I^- of negative traces;
- a definite clause background theory B.

Find: a process model $H \in \mathcal{H}$ such that

- for all $i^+ \in I^+$, $M(B \cup i^+) \models H$;
- for all $i^- \in I^-$, $M(B \cup i^-) \not\models H$;

If $M(B \cup i) \models C$ we say that IC C *covers* the trace i and if $M(B \cup i) \not\models C$ we say that C *rules out* the trace i.

In order to solve the problem, we propose the algorithm DPML (Declarative Process Model Learner) that is an adaptation of ICL [6].

DPML performs a covering loop (function DPML, Figure 1) in which negative interpretations are progressively ruled out and removed from the set I^-. At each

```
function DPML(I+, I−, B)
initialize H := ∅
do
    C := FindBestIC(I+, I−, B)
    if C ≠ ∅ then
        add C to H
        remove from I− all the traces that are false for C
while C ≠ ∅ and I− is not empty
return H

function FindBestIC(I+, I−, B)
initialize Beam := {false ← true}
initialize BestIC := ∅
while Beam is not empty do
    initialize NewBeam := ∅
    for each IC C in Beam do
        for each refinement Ref of C do
            if Ref is better than BestIC then BestIC := Ref
            if Ref is not to be pruned then
                add Ref to NewBeam
                if size of NewBeam > MaxBeamSize then
                    remove worst clause from NewBeam
    Beam := NewBeam
return BestIC
```

Fig. 1. DPML learning algorithm

iteration of the loop a new IC is added to the theory. Each IC rules out some negative interpretations. The loop ends when I^- is empty or when no IC is found.

The IC to be added in every iteration of the covering loop is returned by the procedure FindBestIC (Figure 1). It looks for an IC by using beam search with $p(\ominus|\overline{C})$ as the heuristic function, where $p(\ominus|\overline{C})$ is the probability that an input trace is negative given that is ruled out by the IC C. This heuristic is computed as the number of ruled out negative traces over the total number of ruled out traces (positive and negative). Thus we look for formulas that cover as many positive traces as possible and rule out as many negative traces as possible. The search starts from the IC $false \leftarrow true$ that rules out all the negative traces but also all the positive traces and gradually refines that clause in order to make it more general. Even if the heuristic value of $false \leftarrow true$ is $p(\ominus)$, i.e. the fraction of negative traces in the training set, this IC is initially assigned an heuristic of 0 so that it is not considered better of any other IC. $MaxBeamSize$ is a user-defined constant storing the maximum size of the beam.

The heuristic of each generated refinement is compared with the one of the best IC found so far and, if the value is higher, the best IC is updated. At the end of the refinement cycle, the best IC found so far is returned.

DPML differs from ICL in three respects: we use a different testing procedure, a different refinement operator and a simpler pruning. As regards the refinement operator, $\mathcal{D}$LAB does not allow the possibility of having a conjunction inside negation and it does not allow the deletion of literals from a refinement.

As regards pruning, we do not prune the IC that are not statistically significant but we prune only the refinements that can not become better than the current best clause. We decided to do so because we observed that statistical significance has a low impact on experiments.

4 Experiments

We consider two interaction protocols among agents: an electronic auction protocol [12] and the NetBill protocol [13]. In both cases, we start from a set of ICs describing the protocol, and we randomly generate some traces for the protocol. They are then classified according to the model and are used for learning. For testing, we use a separate set of randomly generated traces.

The first protocol we consider is a sealed bid auction where the auctioneer communicates the bidders the opening of the auction, the bidders answer with bids over the good and then the auctioneer communicates the bidders whether they have won or lost the auction.

The protocol is described by the following ICs [14].

$$\begin{aligned} & bid(B, A, Quote, TBid) \\ \rightarrow & \exists(openauction(A, B, TEnd, TDL, TOpen), \\ & TOpen < TBid, TBid < TEnd) \end{aligned} \tag{3}$$

This IC states that if a bidder sends the auctioneer a *bid*, then there must have been an *openauction* message sent before by the auctioneer and such that the bid has arrived in time (before $TEnd$).

$$\begin{aligned} & openauction(A, B, TEnd, TDL, TOpen), \\ & bid(B, A, Quote, TBid), \\ & TOpen < TBid \\ \rightarrow & \exists(answer(A, B, lose, Quote, TLose), \\ & TLose < TDL, TEnd < TLose) \\ \vee & \exists(answer(A, B, win, Quote, TWin), \\ & TWin < TDL, TEnd < TWin) \end{aligned} \tag{4}$$

This IC states that if there is an *openauction* and a valid *bid*, then the auctioneer must answer with either *win* or *lose* after the end of the bidding time ($TEnd$) and before the deadline (TDL).

$$\begin{aligned} & answer(A, B, win, Quote, TWin) \\ \rightarrow & \forall\neg(answer(A, B, lose, Quote, TLose), TWin < TLose) \end{aligned} \tag{5}$$

$$\begin{aligned} & answer(A, B, lose, Quote, TLose) \\ \rightarrow & \forall\neg(answer(A, B, win, Quote, TWin), TLose < TWin) \end{aligned} \tag{6}$$

These two ICs state that the auctioneer can not answer both *win* and *lose* to the same bidder.

A graphical representation of the protocol is shown in Figure 2.

The traces have been generated in the following way: the first message is always *openauction*, the following messages are generated randomly between *bid* and *answer*. For *answer*, *win* and *lose* are selected randomly with equal probability. The bidders and auctioneer are always the same. The times are selected randomly from 2 to 10. Once a trace is generated, it is tested with the above ICs. If the trace satisfies all the ICs it is added to the set of positive traces, otherwise it is added to the set of negative traces. This process is repeated until 500 positive and 500 negative traces are generated for length 3, 4, 5 and 6. Thus overall there are 2000 positive traces and 2000 negative traces.

NetBill is a security and transaction protocol optimized for the selling and delivery of low-priced information goods, such as software or journal articles, across the Internet. The protocols involves three parties: the customer, the merchant and the NetBill server. Here is an outline of the NetBill protocol (see Figure 3):

1. the customer requests a price for a good from the merchant;
2. the merchant answers with a price for the good;
3. the customer can accept the offer, refuse it or make another request to the merchant, thus initiating a new negotiation and going back to step 2;
4. if the customer accepts the offer, it tells it to the merchant;
5. the merchant delivers the good to the customer encrypted with key K;

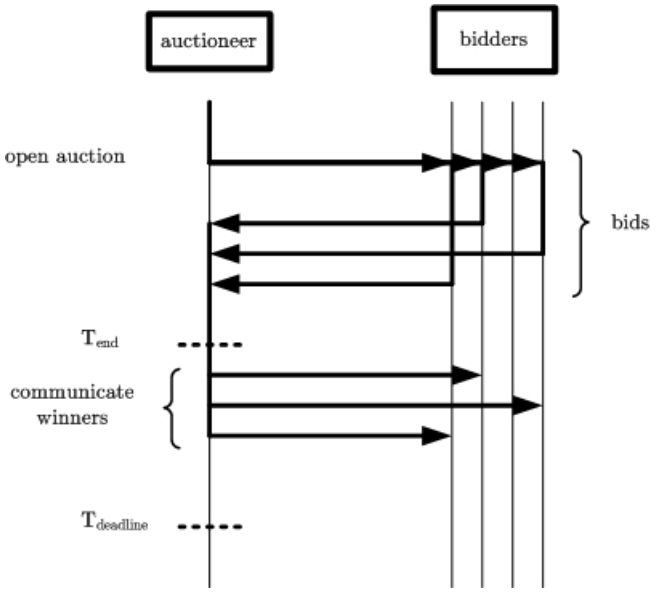


Fig. 2. Sealed bid auction protocol

6. the customer prepares an electronic purchase order (EPO) digitally signed by her and sends it to the merchant;
7. the merchant countersigns the EPO and sends it and the value of K to the NetBill server;
8. the NetBill server checks the signature and counter-signature on the EPO. If customer's account contains enough funds, the NetBill server transfers the price from the customer's account to the merchant's account. The NetBill server then prepares a signed receipt that includes the value K, and it sends this receipt to the merchant;
9. the merchant records the receipt and forwards it to the customer (who can then decrypt her encrypted goods).

The NetBill protocol is represented using 19 ICs [14]. One of them is

$$\begin{aligned} & request(C, M, good(G, Q), NNeg, TReq), \\ & present(M, C, good(G, Q), NNeg, TP), TReq \leq TP \\ \rightarrow & \exists(accept(C, M, good(G, Q), TA), TP \leq TA) \\ \vee & \exists(refuse(C, M, good(G, Q), TRef), TP \leq TRef) \\ \vee & \exists(request(C, M, good(G, Q1), NNeg1, TReq1), TP \leq TReq1) \end{aligned} \tag{7}$$

This IC states that if there has been a *request* from the customer to the merchant and the merchant has answered with the same price, then the customer should either *accept* the offer, *refuse* the offer or start a new negotiation with a *request*.

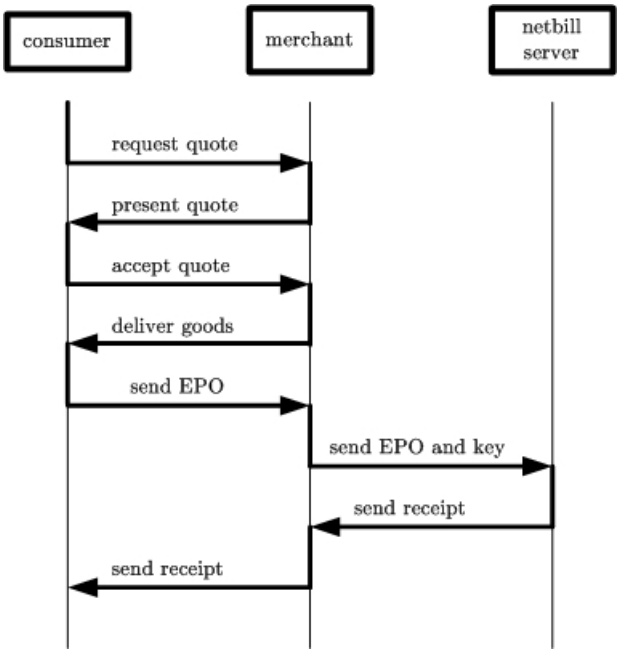


Fig. 3. NetBill transaction protocol

The traces have been generated randomly in two stages: first the negotiation phase is generated and then the transaction phase. In the negotiation phase, we add to the end of the trace a *request* or *present* message with its arguments randomly generated with two possible values for Q (quote). The length of the negotiation phase is selected randomly between 2 and 5. After the completion of the negotiation phase, either an *accept* or a *refuse* message is added to the trace and the transaction phase is entered with probability 4/5, otherwise the trace is closed.

In the transaction phase, the messages *deliver*, *epo*, *epo_and_key*, *receipt* and *receipt_client* are added to the trace. With probability 1/4 a message from the whole trace is then removed.

Once a trace has been generated, it is classified with the ICs of the correct model and assigned to the set of positive or negative traces depending on the result of the test. The process is repeated until 2000 positive traces and 2000 negative traces have been generated.

Five training sets have been generated for the auction protocol and five for the NetBill protocol. Then DPML and the α-algorithm [15] have been applied to each of them. The α-algorithm is one of the first process mining algorithms and it induces Petri nets. We used the implementation of it available in the ProM suite [16]. Since the α-algorithm takes as input a single set of traces, we have provided it with the positive traces only.

The language bias that was used for the auction protocol is the following:

- BS contains two sets of instances of each action (open auction, bid, answer win and answer lose), with the instances of each set having the same variables;
- HS contains a P disjunct for open auction, answer win and answer lose and an N disjunct for open auction, answer win and answer lose. The disjuncts for open auction will contain atoms for the predicate *less* that compare each of its time arguments with the times of the literals in the body. The disjuncts for answer will contain atoms for the predicate *less* that compare its time to the time arguments of the literals in the body.

For example, BS will contain
openauction(f,taxi1,TEnd,TDead,T)
and
openauction(f,taxi1,TEnd1,TDead1,T1)
and HS will contain the P conjunction:
{*answer(f,taxi1,lose,taxi2station,Price3,T2), lessp(T,T2), lessp(T1,T2), lessp(TEnd,T2), lessp(TDead,T2), lessp(TEnd1,T2), lessp(TDead1,T2), lessp(T2,TDead), lessp(T2,TDead1)*}
where $lessp(A, B)$ is a predicate that fails if one of its two arguments is not instantiated and is equal to $A < B$ otherwise. In this way, if one of its arguments is not instantiated, the disjunct can not be true and the learning algorithm must either add the literal with the variable to the body or remove the *lessp* atom.

The language bias that was used for NetBill is the following:

- BS contains two sets of instances of each action (request, present, accept, deliver, send epo, send epo and key, receipt and receipt to client), with the instances of each set having the same variables;
- HS contains a P disjunct and an N disjunct for each action. The disjuncts will contain atoms for the predicate *less* that compare the time argument of the action with the times of the actions in the body plus atoms for the predicate *equal* for comparing the quote of the action in the head with those in the body.

For example, BS will contain
request(c,m,good(software,Q),T)
and
request(c,m,good(software,Q1),T1)
and HS will contain the P conjunction:
{*request(c,m,good(software,Q2,n1,T2), lessp(T,T2), lessp(T1,T2), lessp(T2,T), lessp(T2,T1), equalp(Q2,Q), equalp(Q2,Q1)*}
where $lessp(A, B)$ is defined as before and $equalp(A, B)$ is false if one of the arguments is not instantiated and is equal to $A =:= B$ otherwise.

The learned theories have been tested on five testing set generated with the same procedure used for the training set but with different seeds for the random functions. For the α-algorithm, the Petri net learned from positive traces only was used to replay the positive and negative test traces. The accuracy is given

by the number of positive traces that are replayed correctly plus the number of negative traces not replayed correctly divided by the total number of test traces.

The average accuracy and the standard deviation of DPML and of the α-algorithm are shown in Table 1. The table shows also the the average number of ICs learned by DPML.

Table 1. Results of the experiments

	DPML			α-algorithm	
Experiment	Av. acc.	St. dev.	Av. # ICs	Av. acc.	St. dev.
Auction	97.00%	3.7%	4	-	-
NetBill	94.65%	2.5%	9	66.81%	0.24%

The average time taken by DPML are 0.435 hours for auction and 1.875 hours for NetBill on a Pentium M 2.00 GHz machine. The average time taken by the α-algorithm is under one minute for both datasets.

The α-algorithm was not applied to the auction protocol since it has no way of testing the satisfaction of the deadline, given that it considers an atomic model of the activities.

As can be seen, DPML outperforms the α-algorithm on NetBill in terms of accuracy, even if at the expense of a high computational cost. In order to find out how scalable is our approach, we run a series of experiments with increasing number of traces, from 500 up to 2000. The execution times on a machine with a Core Duo 1.86 GHz are shown in Figure 4. The graph shows that the execution time increases nearly linearly with the number of traces.

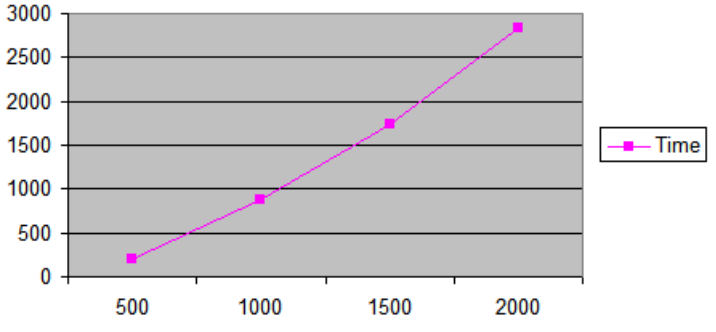


Fig. 4. Scalability of the DPML

5 Related Works

The integrity constraints presented in this paper are inspired to the integrity constraints of the $\mathcal{S}$CIFF language [9,5]. For example, IC (2) would be written in the $\mathcal{S}$CIFF language as

$$\begin{array}{l} \mathbf{H}(a(bob), T) \wedge T < 10 \\ \rightarrow \mathbf{E}(b(alice), T1) \wedge T < T1 \\ \vee \\ \mathbf{EN}(c(mary), T1) \wedge T < T1 \wedge T1 < T + 10 \end{array} \tag{8}$$

where **H** stands for "happened", **E** for "expected to happen" and **EN** for "expected not to happen".

The $\mathcal{S}$CIFF language allows for much more complex ICs than the ones considered in this paper and is equipped with an abductive proof procedure for testing the compliance of a trace. In particular, the $\mathcal{S}$CIFF language allows for the combination of variables with different quantification in the same head disjunct. We focused on a subset for its nice computational properties.

[2] introduced the idea of applying Process Mining to workflow management. The authors propose an approach for inducing a process representation in the form of a directed graph encoding the precedence relationships.

[15] presents the α-algorithm for inducing Petri nets from data and identifies the class of models for which the approach is guaranteed to work. The α-algorithm is based on the discovery of binary relations in the log, such as the "follows" relation.

[4] is a recent work where a process model is induced in the form of a disjunction of special graphs called workflow schemes.

We differ from all of these works in three respects. First, we learn from positive and negative traces, rather than from positive traces only. Second, we use a representation that is declarative rather than procedural as Petri nets are, without sacrificing expressiveness. For example we can model concurrency and synchronization among activities. Third, we can take into account attributes of events, such as in the auction protocol where we check that deadlines are respected.

Other works deal with the learning of integrity constraints, in particular [10,6,17]. However, all of these works learn integrity constraints in the form of clauses, that are less expressive than our formalism.

6 Conclusions and Future Works

We have presented an approach for performing Process Mining by using ILP techniques. The approach introduces a new language that extends the one of disjunctive clauses and that can be used to test the compliance of a trace by simply using a Prolog interpreter. A subsumption relation for the new language is introduced together with a refinement operator.

The similarity with clausal logic allows the use of the ICL algorithm for learning process models. Two experiments have been performed on synthetic data generated from two models of interaction protocols: a sealed bid auction and the NetBill protocol. A good accuracy has been achieved in both experiments. The accuracy on NetBill is higher than the one of the α-algorithm on the same dataset.

In the future, we plan to test the system on real world process logs in order to have a more accurate test of the effectiveness of the approach.

Acknowledgments

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

References

1. Georgakopoulos, D., Hornick, M.F., Sheth, A.P.: An overview of workflow management: From process modeling to workflow automation infrastructure. Distributed and Parallel Databases 3(2), 119–153 (1995)
2. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. In: Schek, H.-J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT 1998. LNCS, vol. 1377, pp. 469–483. Springer, Heidelberg (1998)
3. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data Knowl. Eng. 47(2), 237–267 (2003)
4. Greco, G., Guzzo, A., Pontieri, L., Saccà, D.: Discovering expressive process models by clustering log traces. IEEE Trans. Knowl. Data Eng. 18(8), 1010–1027 (2006)
5. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: An abductive interpretation for open societies. In: Cappelli, A., Turini, F. (eds.) AI*IA 2003. LNCS, vol. 2829, Springer, Heidelberg (2003)
6. De Raedt, L., Van Laer, W.: Inductive constraint logic. In: Zeugmann, T., Shinohara, T., Jantke, K.P. (eds.) ALT 1995. LNCS, vol. 997, Springer, Heidelberg (1995)
7. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, Springer, Heidelberg (2006)
8. Clark, K.L.: Negation as failure. In: Logic and Databases, Plenum Press, New York (1978)
9. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: The SCIFF framework. In: ACM Transactions on Computational Logics (accepted for publication, 2007)
10. Raedt, L.D., Dehaspe, L.: Clausal discovery. Machine Learning 26(2-3), 99–146 (1997)
11. Apt, K.R., Bezem, M.: Acyclic programs. New Generation Comput 9(3/4), 335–364 (1991)
12. Chavez, A., Maes, P.: Kasbah: An agent marketplace for buying and selling goods. In: Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 1996), London, April 1996, pp. 75–90 (1996)
13. Cox, B., Tygar, J., Sirbu, M.: Netbill security and transaction protocol. In: Proceedings of the First USENIX Workshop on Electronic Commerce, New York (July 1995)

14. Socs protocol repository, http://edu59.deis.unibo.it:8079/SOCSProtocolsRepository/jsp/index.jsp
15. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. 16(9), 1128–1142 (2004)
16. Prom framework, http://is.tm.tue.nl/~cgunther/dev/prom/
17. Jorge, A., Brazdil, P.: Integrity constraints in ilp using a monte carlo approach. In: ILP 1996. LNCS, vol. 1314, pp. 229–244. Springer, Heidelberg (1997)

A Refinement Operator Based Learning Algorithm for the $\mathcal{ALC}$ Description Logic

Jens Lehmann[1,⋆] and Pascal Hitzler[2,⋆⋆]

[1] Universität Leipzig, Department of Computer Science
Johannisgasse 26, D-04103 Leipzig, Germany
lehmann@informatik.uni-leipzig.de
[2] Universität Karlsruhe (TH), AIFB Institute
D-76128 Karlsruhe, Germany
hitzler@aifb.uni-karlsruhe.de

Abstract. With the advent of the Semantic Web, description logics have become one of the most prominent paradigms for knowledge representation and reasoning. Progress in research and applications, however, faces a bottleneck due to the lack of available knowledge bases, and it is paramount that suitable automated methods for their acquisition will be developed. In this paper, we provide the first learning algorithm based on refinement operators for the most fundamental description logic $\mathcal{ALC}$. We develop the algorithm from thorough theoretical foundations and report on a prototype implementation.

1 Introduction

The Semantic Web is gaining momentum. Semantic Technologies, based on the same underlying principles, are being applied in adjacent areas such as Software Engineering and Content Management, and industrial interest is rising rapidly. Fundamental to these approaches is the modelling of knowledge by means of ontologies, and the single most popular paradigm for this is by using the Web Ontology Language OWL,[1] which has been recommended by the World Wide Web Consortium (W3C) since 2004.

Progress in research and applications, however, faces a bottleneck due to the lack of available OWL knowledge bases. Considerable effort is therefore currently being invested into developing automated means for the acquisition of ontologies. Most of the currently pursued approaches, however, neglect the expressive power of OWL and are only capable of learning inexpressive ontologies, such as taxonomic hierarchies. As such, they fail by far in leveraging the potential inherent in the expressive features of the Web Ontology Language.

⋆ The first author acknowledges support by the German Federal Ministry of Education and Research (BMBF) under the SoftWiki project (grant 01 ISF02 B).
⋆⋆ The second author acknowledges support by the German Federal Ministry of Education and Research (BMBF) under the SmartWeb project (grant 01 IMD01 B) and by the Deutsche Forschungsgemeinschaft (DFG) under the ReaSem project.
[1] http://www.w3.org/2004/OWL

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 147–160, 2008.

From a logical perspective, OWL is basically an expressive description logic (DL) [1]. It is therefore natural to attempt an adaptation of logic-based approaches to machine-learning for automated ontology acquisition. Inspired by the success of Inductive Logic Programming (ILP), we pursue the transfer of ILP methods [13] to DLs, and in this paper we report on a resulting learning algorithm. Our approach is based on a thorough theoretical analysis of the potential and limitations of refinement operators for DLs. We make the following contributions:

1. development of a refinement operator, which conforms to theoretical findings
2. design of an algorithm handling the unavoidable limitations of this operator
3. provision of a preliminary evaluation

The algorithm was created with extensibility to additional (non-$\mathcal{ALC}$) concept constructors, e.g. number restrictions, in mind. In contrast to previous approaches [6,7,8], we pay more attention to finding simple, non-overfitting solutions of the learning problem.

The paper is structured as follows. After some preliminaries in Section 2, we introduce our refinement operator in Section 3 and show that it conforms to the desired theoretical properties. In Section 4 we extend this refinement operator to a learning algorithm. In Section 5, we report on our prototype implementation and preliminary evaluation. We discuss related work in Section 6 and conclude in Section 7. Proofs had to be omitted for lack of space, but can be found in the technical report [10].

2 Preliminaries

2.1 Description Logics

Description logics represent knowledge in terms of *objects*, *concepts*, and *roles*. Objects correspond to constants, concepts to unary predicates, and roles to binary predicates in first order logic. In DL systems information is stored in a *knowledge base*, which is a set of axioms. It is divided in (at least) two parts: *TBox* (terminology) and *ABox* (assertions). The ABox contains *assertions* about objects. It relates objects to concepts and roles. The TBox describes the *terminology* by relating concepts and roles.

We briefly introduce the $\mathcal{ALC}$ description logic, which is the target language of our learning algorithm and refer to [1] for further background on description logics. As usual in logics, interpretations are used to assign a meaning to syntactic constructs. Let N_I denote the set of objects, N_C denote the set of atomic concepts, and N_R denote the set of roles. An *interpretation* $\mathcal{I}$ consists of a non-empty *interpretation domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$, which assigns to each object $a \in N_I$ an element of $\Delta^{\mathcal{I}}$, to each concept $A \in N_C$ a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and to each role $r \in N_R$ a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Interpretations are extended to concepts as shown in Table 1, and to other elements of a knowledge base in a straightforward way. An interpretation, which satisfies an

Table 1. $\mathcal{ALC}$ syntax and semantics

construct	syntax	semantics
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
role	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
conjunction	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
negation	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
existential	$\exists r.C$	$(\exists r.C)^{\mathcal{I}} = \{a \mid$ $\exists b.(a,b) \in r^{\mathcal{I}}$ and $b \in C^{\mathcal{I}}\}$
universal	$\forall r.C$	$(\forall r.C)^{\mathcal{I}} = \{a \mid$ $\forall b.(a,b) \in r^{\mathcal{I}}$ implies $b \in C^{\mathcal{I}}\}$

axiom (set of axioms) is called a model of this axiom (set of axioms). An $\mathcal{ALC}$ concept is in *negation normal form* if negation only occurs in front of concept names.

If C and D are concepts, then $C \sqsubseteq D$ and $C \equiv D$ are *terminological axioms*. The former axioms are called *inclusions* and the latter *equalities*. An equality whose left hand side is an atomic concept is a *concept definition*.

It is the aim of *inference algorithms* to extract implicit knowledge from a given knowledge base. Standard reasoning tasks include *instance checks*, *retrieval* and *subsumption*. We will only explicitly define the latter. Let C, D be concepts and $\mathcal{T}$ a TBox. *C is subsumed by D*, denoted by $C \sqsubseteq D$, iff for any interpretation $\mathcal{I}$ we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. *$C$ is subsumed by D with respect to $\mathcal{T}$* (denoted by $C \sqsubseteq_{\mathcal{T}} D$) iff for any model $\mathcal{I}$ of $\mathcal{T}$ we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. *$C$ is equivalent to D (with respect to $\mathcal{T}$)*, denoted by $C \equiv D$ ($C \equiv_{\mathcal{T}} D$), iff $C \sqsubseteq D$ ($C \sqsubseteq_{\mathcal{T}} D$) and $D \sqsubseteq C$ ($D \sqsubseteq_{\mathcal{T}} C$). *$C$ is strictly subsumed by D (with respect to $\mathcal{T}$)*, denoted by $C \sqsubset D$ ($C \sqsubset_{\mathcal{T}} D$), iff $C \sqsubseteq D$ ($C \sqsubseteq_{\mathcal{T}} D$) and not $C \equiv D$ ($C \equiv_{\mathcal{T}} D$).

2.2 Learning in Description Logics Using Refinement Operators

Definition 1 (learning problem in description logics). *Let a concept name* `Target`*, a knowledge base $\mathcal{K}$ (not containing* `Target`*), and sets E^+ and E^- with elements of the form* `Target`*(a) ($a \in N_I$) be given. The learning problem is to find a concept C such that* `Target` *does not occur in C and for $\mathcal{K}' = \mathcal{K} \cup \{$*`Target`*$\equiv C\}$ we have $\mathcal{K}' \models E^+$ and $\mathcal{K}' \not\models E^-$.*

By Occam's razor [3] simple solutions of the learning problem are to be preferred over more complex ones, because they have a higher predictive quality. We measure simplicity as the *length* of a concept, which is defined in a straightforward way, namely as the sum of the numbers of concept, role, quantifier, and connective symbols occurring in the concept.

The goal of learning is to find a correct concept with respect to the examples. This can be seen as a search process in the space of concepts. A natural idea

is to impose an ordering on this search space and use operators to traverse it, which is the purpose of *refinement operators*. Intuitively, downward (upward) refinement operators construct specialisations (generalisations) of hypotheses.

A *quasi-ordering* is a reflexive and transitive relation. Let S be a set and $\preceq$ a quasi-ordering on S. In the quasi-ordered space $(S, \preceq)$ a *downward (upward) refinement operator* ρ is a mapping from S to 2^S, such that for any $C \in S$ we have that $C' \in \rho(C)$ implies $C' \preceq C$ ($C \preceq C'$). C' is called a *specialisation* (*generalisation*) of C. Quasi-orderings can be used for searching in the space of concepts. As ordering we can use subsumption. If a concept C subsumes a concept D ($D \sqsubseteq C$), then C covers all examples, which are covered by D, which makes subsumption a suitable order.

Definition 2. *A refinement operator in the quasi-ordered space* $(\mathcal{ALC}, \sqsubseteq_{\mathcal{T}})$ *is called an* $\mathcal{ALC}$ refinement operator.

We need to introduce some notions for refinement operators. A *refinement chain of an* $\mathcal{ALC}$ *refinement operator* ρ *of length* n from a concept C to a concept D is a finite sequence $C_0, C_1, \ldots, C_n$ of concepts, such that $C = C_0, C_1 \in \rho(C_0), C_2 \in \rho(C_1), \ldots, C_n \in \rho(C_{n-1}), D = C_n$. This refinement chain *goes through* E iff there is an i ($1 \leq i \leq n$) such that $E = C_i$. We say that D can be reached from C by ρ if there exists a refinement chain from C to D. $\rho^*(C)$ denotes the set of all concepts, which can be reached from C by ρ. $\rho^m(C)$ denotes the set of all concepts, which can be reached from C by a refinement chain of ρ of length m. If we look at refinements of an operator ρ, we will often write $C \rightsquigarrow_\rho D$ instead of $D \in \rho(C)$. If the used operator is clear from the context it is usually omitted, i.e. we write $C \rightsquigarrow D$.

We will introduce the concept of weak equality of concepts, which is similar to equality of concepts, but takes into account that the order of elements in conjunctions and disjunctions is not important. We say that the concepts C and D are *weakly (syntactically) equal*, denoted by $C \simeq D$ iff they are equal up to permutation of arguments of conjunction and disjunction. Two sets S_1 and S_2 of concepts are weakly equal if for any $C_1 \in S_1$ there is a $C_1' \in S_2$ such that $C_1 \simeq C_1'$ and vice versa. Weak equality of concepts is coarser than equality and finer than equivalence (viewing the equivalence, equality, and weak equality of concepts as equivalence classes). Refinement operators can have certain properties, which can be used to evaluate their usefulness for learning hypothesis.

Definition 3. *An* $\mathcal{ALC}$ *refinement operator* ρ *is called*

- (locally) finite *iff* $\rho(C)$ *is finite for any concept* C.
- redundant *iff there exists a refinement chain from a concept* C *to a concept* D*, which does not go through some concept* E *and a refinement chain from* C *to a concept weakly equal to* D*, which does go through* E.
- proper *iff for all concepts* C *and* D, $D \in \rho(C)$ *implies* $C \not\equiv D$.
- ideal *iff it is finite, complete (see below), and proper.*

An $\mathcal{ALC}$ *downward refinement operator* ρ *is called*

- complete *iff for all concepts* C, D *with* $C \sqsubset_{\mathcal{T}} D$ *we can reach a concept* E *with* $E \equiv C$ *from* D *by* ρ.
- weakly complete *iff for all concepts* $C \sqsubset_{\mathcal{T}} \top$ *we can reach a concept* E *with* $E \equiv C$ *from* $\top$ *by* ρ.

The corresponding notions for upward refinement operators are defined dually.

3 Designing a Refinement Operator

To design a suitable operator, we first look at theoretical limitations. The following theorem from [9] provides a full analysis of the properties of $\mathcal{ALC}$ refinement operators:

Theorem 1 (Property Theorem). *Considering the properties completeness, weak completeness, properness, finiteness, and non-redundancy the following are maximal sets of properties (in the sense that no other of the mentioned properties can be added) of* $\mathcal{ALC}$ *refinement operators (see [9] for details):*

1. {*weakly complete*, *complete*, *finite*}
2. {*weakly complete*, *complete*, *proper*}
3. {*weakly complete*, *non-redundant*, *finite*}
4. {*weakly complete*, *non-redundant*, *proper*}
5. {*non-redundant*, *finite*, *proper*}

Incomplete operators are not interesting, because we may then be unable to find possible solutions, so we can ignore the fifth property combination. We can see from the other combinations, that we can have either finity or properness as a property of an $\mathcal{ALC}$ refinement operator – but not both at the same time. Since we are able to handle infinity quite well, as we will describe in Section 4, we will aim for properness. Our learning algorithm will perform a top-down search, so the fourth combination seems to be desirable, because weak completeness is sufficient in this case. However, an incomplete, but weakly complete operator cannot support some of the features which we consider essential in our learning algorithm. Hence, we decided to use the second combination.

We proceed as follows: First, we define a refinement operator and prove its completeness. We then extend it to a complete and proper operator. Section 4 will show how we handle the problems of redundancy and infinity in the learning algorithm.

For each $A \in N_C$, we define $\text{nb}_{\downarrow}(A) = \{A' \mid A' \in N_C$, there is no $A'' \in N_C$ with $A' \sqsubset_{\mathcal{T}} A'' \sqsubset_{\mathcal{T}} A\}$. $\text{nb}_{\uparrow}(A)$ is defined analogously. In the sequel, we will analyse the refinement operator $\rho_{\downarrow}$ given by:

$$\rho_{\downarrow}(C) = \begin{cases} \{\bot\} \cup \rho'_{\downarrow}(C) & \text{if } C = \top \\ \rho'_{\downarrow}(C) & \text{otherwise} \end{cases}$$

$$\rho'_{\downarrow}(C) = \begin{cases} \emptyset & \text{if } C = \bot \\ \{C_1 \sqcup \dots \sqcup C_n \mid C_i \in M \ (1 \leq i \leq n)\} & \text{if } C = \top \\ \{A' \mid A' \in \mathrm{nb}_{\downarrow}(A)\} \cup \{A \sqcap D \mid D \in \rho'_{\downarrow}(\top)\} & \text{if } C = A \ (A \in N_C) \\ \{\neg A' \mid A' \in \mathrm{nb}_{\uparrow}(A)\} \cup \{\neg A \sqcap D \mid D \in \rho'_{\downarrow}(\top)\} & \text{if } C = \neg A \ (A \in N_C) \\ \{\exists r.E \mid E \in \rho'_{\downarrow}(D)\} \cup \{\exists r.D \sqcap E \mid E \in \rho'_{\downarrow}(\top)\} & \text{if } C = \exists r.D \\ \{\forall r.E \mid E \in \rho'_{\downarrow}(D)\} \cup \{\forall r.D \sqcap E \mid E \in \rho'_{\downarrow}(\top)\} & \text{if } C = \forall r.D \\ \quad \cup \{\forall r.\bot \mid D = A \in N_C \text{ and } \mathrm{nb}_{\downarrow}(A) = \emptyset\} & \\ \{C_1 \sqcap \dots \sqcap C_{i-1} \sqcap D \sqcap C_{i+1} \sqcap \dots \sqcap C_n \mid & \text{if } C = C_1 \sqcap \dots \sqcap C_n \\ \quad D \in \rho'_{\downarrow}(C_i), 1 \leq i \leq n\} & (n \geq 2) \\ \{C_1 \sqcup \dots \sqcup C_{i-1} \sqcup D \sqcup C_{i+1} \sqcup \dots \sqcup C_n \mid & \text{if } C = C_1 \sqcup \dots \sqcup C_n \\ \quad D \in \rho'_{\downarrow}(C_i), 1 \leq i \leq n\} & (n \geq 2) \\ \quad \cup \{(C_1 \sqcup \dots \sqcup C_n) \sqcap D \mid D \in \rho'_{\downarrow}(\top)\} & \end{cases}$$

Fig. 1. Definition of $\rho'_{\downarrow}$

where the operator $\rho'_{\downarrow}$ is defined as in Figure 1. The definition refers to a set M which is inductively defined as follows: All elements in $\{A \mid A \in N_C, \mathrm{nb}_{\uparrow}(A) = \emptyset\}$ (= most general atomic concepts), $\{\neg A \mid A \in N_C, \mathrm{nb}_{\downarrow}(A) = \emptyset\}$ (= negated most specific atomic concepts), and $\{\exists r.\top \mid r \in N_R\}$ are in M. If a concept C is in M, then $\forall r.C$ with $r \in N_R$ is also in M.

Proposition 1. *$\rho_{\downarrow}$ is an $\mathcal{ALC}$ downward refinement operator.*

A distinguishing feature of $\rho_{\downarrow}$ compared to other refinement operators for learning concepts in DLs [2,6] is that it makes use of the subsumption hierarchy. This is useful, since the operator can make use of knowledge contained implicitly in the TBox. Note that $\rho_{\downarrow}$ is infinite. The reason is that the set M is infinite and, furthermore, we put no boundary on the number of elements in the disjunctions, which are refinements of the top concept.

3.1 Completeness of the Operator

To investigate the completeness of the operator, we define a set $S_{\downarrow}$ of $\mathcal{ALC}$ concepts in negation normal form as follows:

Definition 4 ($S_{\downarrow}$). *We define $S_{\downarrow} = S'_{\downarrow} \cup \{\bot\}$, where $S'_{\downarrow}$ is defined as follows:*

1. *If $A \in N_C$ then $A \in S'_{\downarrow}$ and $\neg A \in S'_{\downarrow}$.*
2. *If $r \in N_R$ then $\forall r.\bot \in S'_{\downarrow}$, $\exists r.\top \in S'_{\downarrow}$.*
3. *If $C, C_1, \dots, C_m$ are in $S'_{\downarrow}$ then the following concepts are also in $S'_{\downarrow}$:*
   - *$\exists r.C$, $\forall r.C$, $C_1 \sqcap \dots \sqcap C_m$, and*
   - *$C_1 \sqcup \dots \sqcup C_m$ if for all i $(1 \leq i \leq m)$ C_i is not of the form $D_1 \sqcap \dots \sqcap D_n$ where all D_j $(1 \leq j \leq n)$ are of the form $E_1 \sqcup \dots \sqcup E_p$.*

In $S_\downarrow$, we do not use the $\top$ and $\bot$ symbols directly and we make a restriction on disjunctions, i.e. we do not allow that elements of a disjunction are conjunctions, which in turn only consist of disjunctions. It can be shown that for any $\mathcal{ALC}$ concept C there exists a concept $D \in S_\downarrow$ such that $D \equiv C$.

Lemma 1 ($S_\downarrow$). *For any $\mathcal{ALC}$ concept C there exists a concept $D \in S_\downarrow$ such that $D \equiv C$.*

This allows us to show weak completeness by proving that every element in $S_\downarrow$ can be reached from $\top$ by $\rho_\downarrow$.

Proposition 2 (weak completeness of $\rho_\downarrow$). *$\rho_\downarrow$ is weakly complete.*

Using this, we can prove completeness (again, we refer to [10] for proofs).

Proposition 3. *$\rho_\downarrow$ is complete.*

3.2 Achieving Properness

The operator $\rho_\downarrow$ is not proper, for instance it allows the refinement $\top \rightsquigarrow \exists r.\top \sqcup \forall r.A_1 \quad (\equiv \top)$ where $A_1 \in \mathrm{nb}_\downarrow(\top)$. Indeed, there is no structural subsumption algorithm for $\mathcal{ALC}$ [1], which indicates that it is hard to define a proper operator just by syntactic rewriting rules. One could try to modify $\rho_\downarrow$, such that it becomes proper. Unfortunately, this is likely to lead to incompleteness. Say, we disallow the refinement step just mentioned and consider the following refinement chain:

$$\top \rightsquigarrow \exists r.\top \sqcup \forall r.A_1 \rightsquigarrow \exists r.A_2 \sqcup \forall r.A_1 \quad (A_1, A_2 \in \mathrm{nb}_\downarrow(\top))$$

If we disallow the first step, we would have to ensure that we can reach $\exists r.A_2 \sqcup \forall r.A_1$ from $\top$, otherwise the operator is weakly incomplete. In particular, there can be infinite chains of improper refinements:

$$\top \rightsquigarrow \exists r.\top \sqcup \forall r.A_1 \rightsquigarrow \exists r.(\exists r.\top \sqcup \forall r.A_1) \sqcup \forall r.A_1 \rightsquigarrow \ldots$$

This example illustrates that one would have to allow very complex concepts to be generated as refinements of the top concept, if one wants to achieve weak completeness and properness. So, instead of modifying $\rho_\downarrow$ directly, we allow it to be improper, but consider the closure $\rho_\downarrow^{cl}$ of $\rho_\downarrow$ [2].

Definition 5 ($\rho_\downarrow^{cl}$). *$\rho_\downarrow^{cl}$ is defined as follows: $D \in \rho_\downarrow^{cl}(C)$ iff there exists a refinement chain*

$$C \rightsquigarrow_{\rho_\downarrow} C_1 \rightsquigarrow_{\rho_\downarrow} \ldots \rightsquigarrow_{\rho_\downarrow} C_n = D$$

such that $C \not\equiv_\mathcal{T} D$ and $C_i \equiv C$ for $i \in \{1, \ldots, n-1\}$.

$\rho_\downarrow^{cl}$ is proper by definition. It also inherits the weak completeness of $\rho_\downarrow$, since we do not disallow any refinement steps, but only check whether they are improper. However, it is necessary to show that $\rho_\downarrow^{cl}$ is a meaningful operator, which we will do in the sequel. We already know that $\rho_\downarrow$ is infinite, so it is clear that we

cannot consider all refinements of a concept at a time. Therefore, in practise we will always compute all refinements of a concept up to a given length. A flexible algorithm will allow this length limit to be increased if necessary. Using this technique, an infinite operator can be handled. However, we have to make sure that all refinements up to a given length are computable in finite time. To show this, we need the following lemma.

Lemma 2 ($\rho_\downarrow$ does not reduce length). *$D \in \rho_\downarrow(C)$ implies $|D| \geq |C|$. Furthermore, there are no infinite refinement chains of the form $C_1 \rightsquigarrow_{\rho_\downarrow} C_2 \rightsquigarrow_{\rho_\downarrow} \ldots$ with $|C_1| = |C_2| = \ldots$, i.e. after a finite number of steps we reach a strictly longer concept.*

Proposition 4 (usefulness of $\rho_\downarrow^{cl}$). *For any concept C in negation normal form and any natural number n, the set $\{D \mid D \in \rho_\downarrow^{cl}(C), |D| \leq n\}$ can be computed in finite time.*

Due to Proposition 4 we can use $\rho_\downarrow^{cl}$ in a learning algorithm. For computing $\rho_\downarrow^{cl}$ up to n, it is sufficient to apply the operator until a non-equivalent concept is reached. By a straightforward analysis of the refinement steps, one can show that in the worst case after $O(|N_C| \cdot |C|)$ steps a refinement of greater length will be reached, which bounds the complexity of computing the closure.

4 The Learning Algorithm

So far, we have designed a complete and proper operator. Unfortunately, such an operator has to be redundant and infinite by Theorem 1. We will now describe how to deal with these problems and define the overall learning algorithm.

4.1 Redundancy Elimination

A learning algorithm can be constructed as a combination of a refinement operator, which defines how the search tree can be built, and a search algorithm, which controls how the tree is traversed. The search algorithm specifies which nodes have to be expanded. Whenever the search algorithm encounters a node in the search tree, it could check whether a weakly equal concept already exists in the search tree. If yes, then this node is ignored, i.e. it will not be expanded further and it will not be evaluated. This removes all redundancies, since every concept exists at most once in the search tree.[2] We can still reach any concept, because we have $\rho_\downarrow^{cl}(C) \simeq \rho_\downarrow^{cl}(D)$ if $C \simeq D$, i.e. $\rho_\downarrow^{cl}$ handles weakly equal concepts in the same way. However, this redundancy elimination approach is computationally expensive if performed naively. Hence, we considered it worthwhile to investigate how it can be handled as efficiently as possible.

Note, that we consider weak equality instead of equality here, e.g. we have $A_1 \sqcap A_2 \neq A_2 \sqcap A_1$, but $A_1 \sqcap A_2 \simeq A_2 \sqcap A_1$. In conjunctions and disjunctions, we

[2] More precisely: For each concept there is at most one representative of the equivalence class of weakly syntactical equal concepts in the search tree which is evaluated.

have the problem that we have to guess which pairs of elements are equal to determine whether two concepts are weakly equal. One way to solve this problem is to define an ordering over concepts and require the elements of disjunctions and conjunctions to be ordered accordingly. This eliminates the guessing step and allows to check weak equality in linear time. There are different ways to define a linear order $\preceq$ over $\mathcal{ALC}$ concepts, and we have shown that it is also possible to do it in such a way that deciding $\preceq$ for two concepts is polynomial and transforming a concept in negation normal form to $\preceq$ *ordered negation normal form*, i.e. elements in conjunctions and disjunctions are ordered with respect to $\preceq$, can be done in polynomial time – for brevity we omit the details. It is thus reasonable to assume that every concept occurring in our search tree can be transformed to ordered negation normal form with respect to some linear order over $\mathcal{ALC}$ concepts. We can then maintain an ordered set of concepts occurring in the search tree. Checking weak equality of a concept C with respect to a search tree containing n concepts will then only require $\log n$ comparisons (binary search), where each comparison needs only linear time. Taking into account the complexity of instance checks (PSPACE for $\mathcal{ALC}$, NEXPTIME for $\mathcal{SHOIN}(D)$ and OWL-DL), which we can avoid (compared to an algorithm without redundancy check), redundancy elimination can be considered reasonable.

4.2 Creating a Full Learning Algorithm

Learning concepts in DLs is a search process. In our proposed learning algorithm, the refinement operator $\rho^{cl}_{\downarrow}$ is used for building the search tree, while a heuristics decides which nodes to expand. To define a search heuristics for our learning algorithm, we need some notions to be able to express what we consider a good concept.

Definition 6 (quality). *Let $\mathcal{K}$ be a knowledge base, E^- the set of negative examples, and E^+ the set of positive examples of a learning problem. The* quality *of a concept C is a function, which maps a concept to an element of $\mathcal{Q}$ with $\mathcal{Q} = \{0, \ldots, -|E^-|\} \cup \{tw\}$, defined by $q(C) = tw$ if there is an $e \in E^+$ with $\mathcal{K} \cup \{C\} \not\models e$ and $q(C) = -|\{e \mid e \in E^- \text{and } \mathcal{K} \cup \{C\} \models e\}|$ otherwise.*

The quality of a concept is "tw" if it is too weak, i.e. it does not cover all positive examples. In all other cases, we assign a number n with $n \geq 0$ to a concept, which is the number of negative examples covered.

As mentioned before, we want to tackle the infinity of the operator by considering only refinements up to some length n at a given time. We call n the *horizontal expansion* of a node. It is a node specific upper bound on the length of child concepts, which can be increased dynamically by the algorithm during the learning process. To deal with this, we formally define a *node* in a search tree to be a quadruple (C, n, q, b), where C is an $\mathcal{ALC}$ concept, $n \in \mathbb{N}$ is the *horizontal expansion*, $q \in \mathcal{Q} \cup \{\text{-}\}$ is the *quality* (- stands for non-evaluated quality), and $b \in \{\text{true}, \text{false}\}$ is a boolean marker for the redundancy of a node.

The search heuristics selects the fittest node in the search tree at a given time. We define fitness as a lexicographical order over quality and horizontal expansion.

Definition 7 (fitness). *Let* $N_1 = (C_1, n_1, q_1, b_1)$ *and* $N_2 = (C_2, n_2, q_2, b_2)$ *be two nodes with defined quality* $(q_1, q_2 \neq -, tw)$. N_1 *is* fitter *than* N_2*, denoted by* $N_2 \leq_f N_1$ *iff* $q_2 < q_1$ *or* $q_1 = q_2$ *and* $n_1 \leq n_2$.

Note, that we use horizontal expansion instead of concept length as second criterion, which makes the algorithm more flexible in searching less explored areas of the search space. More sophisticated ways of ordering concepts are also possible, e.g. tradeoffs between quality and horizontal expansion. Such fitness heuristics enable the algorithm to handle noise. The fitness function can be defined independently of the core learing algorithm.

We have now introduced all necessary notions to specify the complete learning algorithm, given in Algorithm 1. *checkRed* is the redundancy check function and *transform* the function to transform a concept to ordered negation normal form.

Algorithm 1. learning algorithm

Input: *horizExpFactor* in]0,1]
1 ST (search tree) is set to the tree consisting only of the root node $(\top, 0, q(\top), \mathit{false})$
2 $\mathit{minHorizExp} = 0$
3 **while** *ST does not contain a correct concept* **do**
4 choose $N = (C, n, q, b)$ with highest fitness in ST
5 expand N up to length $n + 1$, i.e. :
6 **begin**
7 add all nodes $(D, n, -, \mathit{checkRed}(ST, D))$ with $D \in \mathit{transform}(\rho^{cl}_{\downarrow}(C))$ and $|D| = n + 1$ as children of N
8 evaluate created non-redundant nodes
9 change N to $(C, n + 1, q, b)$
10 **end**
11 $\mathit{minHorizExp} = \max(\mathit{minHorizExp}, \lceil \mathit{horizExpFactor} * (n + 1)) \rceil)$
12 **while** *there are nodes with defined quality and horiz. expansion smaller minHorizExp* **do**
13 expand these nodes up to *minHorizExp*
14 Return a correct concept in ST

We see, that the usual expansion in a search algorithm is replaced by a one step horizontal expansion. If we only expand the fittest node, we may not explore large parts of the search space. In order to avoid this, a minimum horizontal expansion factor is used, which specifies that all nodes have to be expanded at least up to this length. This factor allows us to control the tradeoff between expanding only the fittest nodes and exploring other parts of the search space.

Correctness of the algorithm can be shown:

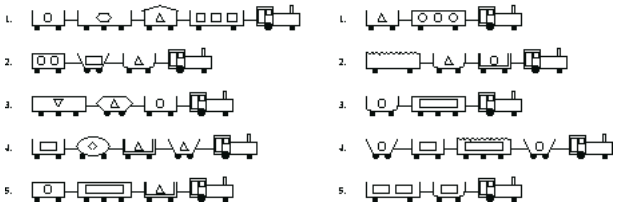


Fig. 2. Michalski trains

Proposition 5 (correctness). *If a learning problem has a solution, then Algorithm 1 terminates and computes a correct solution of the learning problem.*

5 Preliminary Evaluation

We want to illustrate our algorithm using Michalski's trains [12] as an example. The data describes different features of trains, e.g. which cars are appended to a train, whether they are short or long, closed or open, jagged or not, which shapes they contain and how many of them. The positive examples are the trains on the left and the negative examples are the trains on the right. Thus, the task of the learner is to find characteristics of all the left trains, which none of the right trains has. The learning algorithm first explores the concepts $\top$ and `Train`, which cover all examples. Other atomic concepts are too weak to be considered for further exploration. The exploration of the top concept leads to $\exists$`hasCar`.$\top$, which is then expanded to $\exists$`hasCar.Closed`. This covers all positives and two negatives. The heuristic picks this node and extends it to $\exists$`hasCar`.(`Closed`$\sqcap$`Short`), which is a possible (and shortest) solution for the problem.

Doubtless, there is a lack of evaluation standards in ontology learning from examples. In order to overcome this problem, we converted the background knowledge of several existing learning problems to OWL ontologies. Besides the described train problem, we also investigated the problems of learning arches [14], learning poker hands, and understanding the moral reasoning of humans. The two latter examples were taken from the UCI Machine Learning repository[3]. For the poker example, we defined two goals: learning the definition of a pair and of a straight. Similarly, the moral reasoner examples were divided into two learning tasks: the original one, where the intended solution is quite short, and a problem, where we removed an important intermediate concept, such that the smallest possible solution became more complex.

The arch problem is small in terms of size and complexity of the background knowledge. The poker example is larger in terms of size, but still not very complex. The moral reasoner, however, is an expressive ontology, which we derived from a theory given as logic program. The solutions of the examples cover a range of different concept constructors and are of varying length and complexity.

[3] http://www.ics.uci.edu/~mlearn/MLRepository.html

For all test runs we used a (non-optimised) horizontal expansion factor of 0.6. As a reasoner we used Pellet[4] (version 1.4RC1), which was connected to the learning program using the DIG 1.1 interface[5] on a 1.4 GHz CPU machine. The only system we could use for comparison is YinYang [8]. The system in [5] is no longer available and the approach in [2] was not fully implemented.

Table 2 summarises the results we obtained. In all cases, our implementation – called DL-Learner – was able to learn the shortest correct definition (which coincides with the intended solution of these problems). YinYang produces longer solutions and could not solve the second poker problem (it produces an error after trying to compute most specific concepts for some time). It generated an incorrect answer for both moral reasoner problems. The percentage of time spent for reasoner requests increases with the complexity and size of background knowledge and the number of examples (it is $> 99\%$ for the moral reasoner problems), which shows that minimizing the number of reasoner requests, e.g. by redundancy elimination, is an important issue.

Table 2. Evaluation results for DL-Learner and YinYang

problem	axioms, concepts, roles objects, examples	DL-Learner			YinYang		
		runtime	length	correct	runtime	length	correct
trains	252, 8, 5, 50, 10	1.1s	5	100%	2.3s	8	100%
arches	71, 6, 5, 19, 5	4.6s	9	100%	1.5s	23	100%
moral (simple)	2176, 43, 4, 45, 43	17.7s	3	100%	205.3s	69	67.4%
moral (complex)	2107, 40, 4, 45, 43	88.1s	8	100%	181.4s	70	69.8%
poker (pair)	1335, 2, 6, 311, 49	7.7s	5	100%	17.1s	43	100%
poker (straight)	1419, 2, 6, 347, 55	35.6s	11	100%	-	-	-

6 Related Work

An interesting paper close to our work is [2]. It suggests a refinement operator for the $\mathcal{ALER}$ description logic. They also investigate some theoretical properties of refinement operators. As we have done with the design of $\rho_{\downarrow}$, they favour the use of a downward refinement operator to enable a top-down search. The authors use $\mathcal{ALER}$ *normal form* (see the paper for a detailed description), which is easier to handle than negation normal form, because $\mathcal{ALER}$ is not closed under boolean operations. As a consequence, they obtain a simpler refinement operator, for which it is not clear how it could be extended to more expressive DLs. Our operator, in contrast, lends itself much easier to such extensions. We also deal quite differently with infinity, show how the subsumption hierarchy of atomic concepts can be used, and describe how redundancy can be avoided efficiently.

A second area of ongoing related work is described in [6,7,8]. They take a different approach for solving the learning problem by using approximated MSCs

[4] http://pellet.owldl.com
[5] http://dl.kr.org/dig/

(most specific concepts). A problem of these algorithms is that they tend to produce unnecessarily long concepts. One reason is that MSCs for $\mathcal{ALC}$ and more expressive languages do not exist and hence can only be approximated. Previous work [4,5] in learning in DLs has mostly focused on approaches using least common subsumers, which face this problem to an even larger extent.

In our approach, we also cannot guarantee that we obtain the shortest possible solution of a learning problem. However, the learning algorithm was carefully designed to produce short solutions. The produced solutions will be close to the shortest solution in negation normal form and, thus, overfitting is unlikely.

Another related area of research are approaches for learning in the hybrid language AL-log [11], which combines $\mathcal{ALC}$ with the function free Horn clause language Datalog.

7 Conclusions and Further Work

To the best of our knowledge, our work presents the first refinement operator based learning algorithm for expressive DLs which are closed under boolean operations.[6] It is based on thorough theoretical investigations concerning the potential of using refinement operators for DLs, and we have shown formally that our operator satisfies the desirable properties which are achievable. We also showed how the problems of redundancy and infinity can be solved in a satisfiable manner, allowing us to specify a learning algorithm which we proved to be correct. We implemented the algorithm and an evaluation showed the feasibility of our approach.

Future work will focus on increasing the expressiveness of the learned language, integrating the learning algorithm in an ontology editor, creating benchmark datasets, and testing on real world data sets.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
2. Badea, L., Nienhuys-Cheng, S.-H.: A refinement operator for description logics. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 40–59. Springer, Heidelberg (2000)
3. Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.K.: Occam's razor. In: Shavlik, J.W., Dietterich, T.G. (eds.) Readings in Machine Learning, pp. 201–204. Morgan Kaufmann, San Francisco (1990)
4. Cohen, W.W., Borgida, A., Hirsh, H.: Computing least common subsumers in description logics. In: Proceedings of the Tenth National Conference on Artificial Intelligence, pp. 754–760. AAAI Press, Menlo Park (1993)

[6] To be precise, [8] also uses refinement operators, but not as centrally as in our approach – see Section 6.

5. Cohen, W.W., Hirsh, H.: Learning the CLASSIC description logic: Theoretical and experimental results. In: Doyle, J., Sandewall, E., Torasso, P. (eds.) Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning, may 1994, pp. 121–133. Morgan Kaufmann, San Francisco (1994)
6. Esposito, F., Fanizzi, N., Iannone, L., Palmisano, I., Semeraro, G.: Knowledge-intensive induction of terminologies from metadata. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 441–455. Springer, Heidelberg (2004)
7. Fanizzi, N., Iannone, L., Palmisano, I., Semeraro, G.: Concept formation in expressive description logics. In: ECML 2004. LNCS (LNAI), vol. 3201, Springer, Heidelberg (2004)
8. Iannone, L., Palmisano, I.: An algorithm based on counterfactuals for concept learning in the semantic web. In: Proceedings of the 18th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Bari, Italy, June 2005, pp. 370–379 (2005)
9. Lehmann, J., Hitzler, P.: A refinement operator based learning algorithm for the $\mathcal{ALC}$ description logic. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS (LNAI), vol. 4894, pp. 147–160. Springer, Heidelberg (2007)
10. Lehmann, J., Hitzler, P.: A refinement operator based learning algorithm for the $\mathcal{ALC}$ description logic. In: Technical report, University of Leipzig (2007), http://www.jens-lehmann.org
11. Lisi, F.A., Malerba, D.: Ideal refinement of descriptions in AL-log. In: Horváth, T., Yamamoto, A. (eds.) ILP 2003. LNCS (LNAI), vol. 2835, pp. 215–232. Springer, Heidelberg (2003)
12. Michalski, R.S.: Pattern recognition as rule-guided inductive inference. IEEE Transactions on Pattern Analysis and Machine Intelligence 2(4), 349–361 (1980)
13. Nienhuys-Cheng, S.-H., de Wolf, R. (eds.): Foundations of Inductive Logic Programming. LNCS. Springer, Heidelberg (1997)
14. Winston, P.: Learning structural descriptions from examples. In: Winston, P. (ed.) The Psychology of Computer Vision, pp. 157–209. McGraw-Hill, New York (1975)

Foundations of Refinement Operators for Description Logics

Jens Lehmann[1,⋆] and Pascal Hitzler[2,⋆⋆]

[1] Universität Leipzig, Department of Computer Science
Johannisgasse 26, D-04103 Leipzig, Germany
lehmann@informatik.uni-leipzig.de
[2] Universität Karlsruhe (TH), AIFB Institute
D-76128 Karlsruhe, Germany
hitzler@aifb.uni-karlsruhe.de

Abstract. In order to leverage techniques from Inductive Logic Programming for the learning in description logics (DLs), which are the foundation of ontology languages in the Semantic Web, it is important to acquire a thorough understanding of the theoretical potential and limitations of using refinement operators within the description logic paradigm. In this paper, we present a comprehensive study which analyses desirable properties such operators should have. In particular, we show that ideal refinement operators in general do not exist, which is indicative of the hardness inherent in learning in DLs. We also show which combinations of desirable properties are theoretically possible, thus providing an important step towards the definition of practically applicable operators.

1 Introduction

With the advent of the Semantic Web and Semantic Technologies, ontologies are becoming one of the most prominent paradigms for knowledge representation and reasoning. However, recent progress in the field faces a lack of available ontologies due to the fact that engineering such ontologies constitutes a considerable investment of resources. Methods for the automated acquisition of ontologies are therefore required. In this article, we develop theoretical foundations for the creation of such methods.

In 2004, the World Wide Web Consortium (W3C) recommended the Web Ontology Language OWL[1] as a standard for modelling ontologies on the Web. In the meantime, many studies and applications using OWL have been reported in research, many of which go beyond Internet usage and employ the power of

⋆ The first author acknowledges support by the German Federal Ministry of Education and Research (BMBF) under the SoftWiki project (grant 01 ISF02 B).
⋆⋆ The second author acknowledges support by the German Federal Ministry of Education and Research (BMBF) under the SmartWeb project (grant 01 IMD01 B) and by the Deutsche Forschungsgemeinschaft (DFG) under the ReaSem project.
[1] http://www.w3.org/2004/OWL/

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 161–174, 2008.

ontological modelling in other fields like software engineering, knowledge management, and cognitive systems.

In essence, OWL coincides with the description logic $\mathcal{SHOIN}(\mathrm{D})$ and is thus a knowledge representation formalism based on first-order logic. In order to leverage machine-learning approaches for the acquisition of OWL ontologies, it is therefore required to develop methods and tools for the learning in description logics. To date, only few investigations have been carried out on this topic, which can be attributed to the fact that description logics (DLs) have only recently become a major paradigm in knowledge representation and reasoning.

In this paper, we investigate the applicability of methods from Inductive Logic Programming (ILP) for the learning in description logic knowledge bases. We are motivated by the success of ILP methods and believe that similar results can be achieved for DLs.

Central to the usual ILP approach are the so-called *refinement operators* which are used to traverse the search space, and many approaches indeed hinge on the definition of a suitable such operator. Theoretical investigations on ILP refinement operators have identified desirable properties for them to have, which impact on their performance. These properties thus provide guidelines for the definition of suitable operators. It turns out, however, that for hard learning settings there are theoretical limitations on the properties a refinement operator can have. A corresponding general analysis therefore provides a clear understanding of the difficulties inherent in a learning setting, and also allows to derive directions for researching suitable operators.

In this paper we therefore give a full analysis of properties of refinement operators for description logics. To the best of our knowledge, such a complete analysis has not been done before, but the need for this investigation was expressed in [6,7]. The main contribution of this article is to derive a fundamental theorem about properties of refinement operators in DLs. This can serve as the foundation for the design of concrete refinement operators, which are used for induction – with potential applications in other areas of Machine Learning like clustering and data mining.

The paper is structured as follows. In Section 2 we will give a brief introduction to description logics. Section 3 formally describes the learning problem in description logics. Refinement operators and their properties are introduced. The main section is Section 4, which contains the results we obtained. We will fully analyse all combinations of interesting properties of refinement operators. This means we will show which combinations of properties are possible, i.e. for which combinations a refinement operator with these properties exists. We make only basic assumptions with respect to the description logic we are looking at, to cover as many description logics as possible. In Section 5 we discuss related work, in particular the relation to refinement operators in the area of traditional Inductive Logic Programming. Finally, in Section 6 we summarise our work and draw conclusions.

Some proofs are omitted for lack of space. They can be found in a separate technical report [12].

2 Description Logics

Description logics represent knowledge in terms of *objects*, *concepts*, and *roles*. Objects correspond to constants, concepts to unary predicates, and roles to binary predicates in first order logic. In description logic systems information is stored in a *knowledge base*, which is a set of axioms. It is divided in (at least) two parts: *TBox* and *ABox*. The ABox contains *assertions* about objects. It relates objects to concepts and roles. The TBox describes the *terminology* by relating concepts and roles.

We briefly introduce the $\mathcal{ALC}$ description logic, which is the target language of our learning algorithm and refer to [1] for further background on description logics. Syntax and semantic of $\mathcal{ALC}$ concept constructors is shown in Table 1. As usual in logics, interpretations are used to assign a meaning to syntactic constructs. Let N_I denote the set of objects, N_C denote the set of atomic concepts, and N_R denote the set of roles. An *interpretation* $\mathcal{I}$ consists of a non-empty *interpretation domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$, which assigns to each object $a \in N_I$ an element of $\Delta^{\mathcal{I}}$, to each concept $A \in N_C$ a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and to each role $r \in N_R$ a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Interpretations are extended to concepts as shown in Table 1, and to other elements of a knowledge base in a straightforward way. An interpretation, which satisfies an axiom (set of axioms) is called a model of this axiom (set of axioms). An $\mathcal{ALC}$ concept is in *negation normal form* if negation only occurs in front of concept names. $\mathcal{SHOIN}(D)$, the description language corresponding to OWL (to be precise it corresponds to the dialect OWL DL), is an extension of $\mathcal{ALC}$. The results presented in this paper are general, i.e. they hold for all reasonable expressive description logics. (The criteria these languages have to fulfil are described later in Definition 3.)

It is the aim of *inference algorithms* to extract implicit knowledge from a given knowledge base. Standard reasoning tasks include *instance checks*, *retrieval* and *subsumption*. We will only explicitly define the latter. Let C, D be concepts and $\mathcal{T}$ a TBox. C *is subsumed by* D, denoted by $C \sqsubseteq D$, iff for any interpretation $\mathcal{I}$ we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. C *is subsumed by* D *with respect to* $\mathcal{T}$ (denoted by $C \sqsubseteq_{\mathcal{T}} D$)

Table 1. $\mathcal{ALC}$ syntax and semantics

construct	syntax	semantics
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
role	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
conjunction	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
negation	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
existential	$\exists r.C$	$(\exists r.C)^{\mathcal{I}} = \{a \mid \exists b.(a,b) \in r^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}$
universal	$\forall r.C$	$(\forall r.C)^{\mathcal{I}} = \{a \mid \forall b.(a,b) \in r^{\mathcal{I}} \text{ implies } b \in C^{\mathcal{I}}\}$

iff for any model $\mathcal{I}$ of $\mathcal{T}$ we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. *$C$ is equivalent to D (with respect to $\mathcal{T}$)*, denoted by $C \equiv D$ ($C \equiv_{\mathcal{T}} D$), iff $C \sqsubseteq D$ ($C \sqsubseteq_{\mathcal{T}} D$) and $D \sqsubseteq C$ ($D \sqsubseteq_{\mathcal{T}} C$). *$C$ is strictly subsumed by D (with respect to $\mathcal{T}$)*, denoted by $C \sqsubset D$ ($C \sqsubset_{\mathcal{T}} D$), iff $C \sqsubseteq D$ ($C \sqsubseteq_{\mathcal{T}} D$) and not $C \equiv D$ ($C \equiv_{\mathcal{T}} D$).

3 Learning in Description Logics Using Refinement Operators

In this section we will briefly describe the learning problem in description logics. The process of learning in logics, i.e. finding logical explanations for given data, is also called *inductive reasoning*. In a very general setting this means that we have a logical formulation of background knowledge and some observations. We are then looking for ways to extend the background knowledge such that we can explain the observations, i.e. they can be deduced from the modified knowledge. More formally we are given background knowledge B, positive examples E^+, negative examples E^- and want to find a hypothesis H such that from H together with B the positive examples follow and the negative examples do not follow. It is not required that the same logical formalism is used for background knowledge, examples, and hypothesis, but often this is the case.

Definition 1 (learning problem in description logics). *Let a concept name* `Target`*, a knowledge base $\mathcal{K}$ (not containing* `Target`*), and sets E^+ and E^- with elements of the form* `Target`*(a) ($a \in N_I$) be given. The learning problem is to find a concept C such that* `Target` *does not occur in C and for $\mathcal{K}' = \mathcal{K} \cup \{\texttt{Target} \equiv C\}$ we have $\mathcal{K}' \models E^+$ and $\mathcal{K}' \not\models E^-$.*

The goal of learning is to find a correct concept with respect to the examples. This can be seen as a search process in the space of concepts. A natural idea is to impose an ordering on this search space and use operators to traverse it. This idea is well-known in Inductive Logic Programming [18], where refinement operators are widely used to find hypotheses. Intuitively, downward (upward) refinement operators construct specialisations (generalisations) of hypotheses.

Definition 2. *A* quasi-ordering *is a reflexive and transitive relation. Let S be a set and $\preceq$ a quasi-ordering on S. In the quasi-ordered space $(S, \preceq)$ a* downward (upward) refinement operator *ρ is a mapping from S to 2^S, such that for any $C \in S$ we have that $C' \in \rho(C)$ implies $C' \preceq C$ ($C \preceq C'$). C' is called a* specialisation *(*generalisation*) of C.*

This idea can be used for searching in the space of concepts. As ordering we can use subsumption. (Note that the subsumption relation $\sqsubseteq$ is a quasi-ordering.) If a concept C subsumes a concept D ($D \sqsubseteq C$), then C will cover all examples, which are covered by D. This makes subsumption a suitable order for searching in concepts. In this section we will analyse refinement operators for concepts with respect to subsumption and a description language $\mathcal{L}$, which we will call $\mathcal{L}$ refinement operators in the sequel.

Definition 3. *Let $\mathcal{L}$ be a description language, which allows to express $\top$, $\bot$, conjunction, disjunction, universal quantification, and existential quantification. A refinement operator in the quasi-ordered space $(\mathcal{L}, \sqsubseteq)$ is called an* $\mathcal{L}$ refinement operator*.*

We need to introduce some notions for refinement operators.

Definition 4. *A* refinement chain of an $\mathcal{L}$ refinement operator ρ of length n *from a concept C to a concept D is a finite sequence $C_0, C_1, \ldots, C_n$ of concepts, such that $C = C_0, C_1 \in \rho(C_0), C_2 \in \rho(C_1), \ldots, C_n \in \rho(C_{n-1}), D = C_n$. This refinement chain* goes through *E iff there is an i $(1 \leq i \leq n)$ such that $E = C_i$. We say that D can be reached from C by ρ if there exists a refinement chain from C to D. $\rho^*(C)$ denotes the set of all concepts, which can be reached from C by ρ. $\rho^m(C)$ denotes the set of all concepts, which can be reached from C by a refinement chain of ρ of length m.*

Definition 5. *A concept C is a* downward cover *of a concept D iff $C \sqsubset D$ and there does not exist a concept E with $C \sqsubset E \sqsubset D$. A concept C is an* upward cover *of a concept D iff $D \sqsubset C$ and there does not exist a concept E with $D \sqsubset E \sqsubset C$.*

If we look at refinements of an operator ρ we will often write $C \rightsquigarrow_\rho D$ instead of $D \in \rho(C)$. If the used operator is clear from the context it is usually omitted, i.e. we write $C \rightsquigarrow D$.

We will introduce the concept of weak equality of concepts, which is similar to equality of concepts, but takes into account that the order of elements in conjunctions and disjunctions is not important. By equality of two concepts we mean that the concepts are syntactically equal. Equivalence of two concepts means that the concepts are logically equivalent (see preliminaries). Weak equality of concepts is coarser than equality and finer than equivalence (viewing the equivalence, equality, and weak equality of concepts as equivalence classes).

Definition 6. *We say that the concepts C and D are* weakly (syntactically) equal*, denoted by $C \simeq D$ iff they are equal up to permutation of arguments of conjunction and disjunction. Two sets S_1 and S_2 of concepts are weakly equal if for any $C_1 \in S_1$ there is a $C_1' \in S_2$ such that $C_1 \simeq C_1'$ and vice versa.*

Refinement operators can have certain properties, which can be used to evaluate their usefulness for learning hypotheses. These properties are what we investigate in this paper.

Definition 7. *An $\mathcal{L}$ refinement operator ρ is called*

- (locally) finite *iff $\rho(C)$ is finite for any concept C.*
- (syntactically) redundant *iff there exists a refinement chain from a concept C to a concept D, which does not go through some concept E and a refinement chain from C to a concept weakly equal to D, which does go through E.*

- proper *iff for all concepts C and D, $D \in \rho(C)$ implies $C \not\equiv D$.*
- ideal *iff it is finite, complete (see below), and proper.*

An $\mathcal{L}$ downward refinement operator ρ is called

- complete *iff for all concepts C and D with $C \sqsubset D$ we can reach a concept E with $E \equiv C$ from D by ρ.*
- weakly complete *iff for all concepts C with $C \sqsubset \top$ we can reach a concept E with $E \equiv C$ from $\top$ by ρ.*
- minimal *iff for all C, $\rho(C)$ contains only downward covers and all its elements are incomparable with respect to $\sqsubseteq$.*

The corresponding notions for upward refinement operators are defined dually.

4 Analysing the Properties of Refinement Operators

In this section we will analyse the properties of refinement operators in description logics. In particular, we are interested in seeing which desired properties can be combined in a refinement operator and which properties are impossible to combine. This is interesting for two reasons: The first one is that this gives us a good impression of how hard (or easy) it is to learn concepts. The second reason is that this can also serve as a practical guide for designing refinement operators. Knowing the theoretical limits allows the designer of a refinement operator to focus on achieving the best possible properties.

$\mathcal{ALC}$ refinement operators have been designed in [6,9]. However, a full theoretical analysis for DL refinement operators has not been done to the best of our knowledge (not even for a specific language). Therefore all propositions in this section are new unless explicitly mentioned otherwise.

As a first property we will briefly analyse minimality of $\mathcal{L}$ refinement operators, in particular the existence of upward and downward covers in $\mathcal{ALC}$. It is not immediately obvious that e.g. downward covers exist in $\mathcal{ALC}$, because it could be the case that for any concept C and D with $C \sqsubset D$ one can always construct a concept E with $C \sqsubset E \sqsubset D$. However, it is possible to show that downward covers do exist.

Proposition 1. *Downward (upward) covers of $\top$ ($\bot$) exist in $\mathcal{ALC}$.*

Example 1. The following is a downward cover of the $\top$ concept:

$$\bigsqcup_{r \in N_R} \exists r.\top \sqcup \bigsqcup_{A \in N_C} A$$

This means that non-trivial minimal operators, i.e. operators which do not map every concept to the empty set, can be constructed. However, minimality of refinement steps is not a directly desired goal in general. Minimal operators are in some languages more likely to lead to overfitting, because they may not produce sufficient generalisation leaps. This is a problem in languages which are closed under boolean operations, i.e. $\mathcal{ALC}$ and more expressive languages.

Indeed, the following result suggests that minimality may not play a central role for DL refinement operators, as it is incompatible with (weak) completeness. A weaker result was claimed to hold, but not proved, in [4]. We formulate our result for the description logic $\mathcal{AL}$, the concepts of which are inductively defined as follows: $\top$, $\bot$, $\exists r.\top$, A, $\neg A$ with $A \in N_C$, $r \in N_R$ are $\mathcal{AL}$ concepts. If C and D are $\mathcal{AL}$ concepts, then $C \sqcap D$ is an $\mathcal{AL}$ concept. If C is an $\mathcal{AL}$ concept and r a role, then $\forall r.C$ is an $\mathcal{AL}$ concept.

Proposition 2. *There exists no minimal and weakly complete $\mathcal{AL}$ downward refinement operator.*

In the sequel we analyse desired properties of $\mathcal{L}$ refinement operators: completeness, properness, finiteness, and non-redundancy. We show several positive and negative results, which together yield a full analysis of these properties.

Proposition 3. *For any language $\mathcal{L}$, which satisfies the conditions stated in Definition 3, there exists a complete and finite $\mathcal{L}$ refinement operator.*[2]

In particular, the following operator can be shown to be complete and finite for any language $\mathcal{L}$ we consider:

$$\rho(C) = \{C \sqcap \top\} \cup \{D \mid |D| \leq (\text{number of } \top \text{ occurrences in } C) \text{ and } D \sqsubset C\}$$

Of course, it is obvious that the operator used to prove Proposition 3 is not useful in practise, since it merely generates concepts without paying attention to efficiency. However, in [11], we have developed a complete and finite operator, which was integrated into the Genetic Programming framework and shown to be useful in a preliminary evaluation.

Let it be noted that in many scenarios it appears to be quite difficult to design a good complete and finite refinement operator. The reason is that finiteness can only be achieved by using non-proper refinement steps. We will now show that it is impossible to define a complete, finite, and proper refinement operator. Such operators are known as ideal and their non-existence indicates that learning concepts in sufficiently expressive description logics is hard.

Proposition 4. *For any language $\mathcal{L}$, which satisfies the conditions stated in Definition 3, there does not exist any ideal $\mathcal{L}$ refinement operator.*

We will give a proof sketch: By contradiction, we assume that there exists an ideal downward refinement operator ρ. We further assume that there is a role $r \in N_R$. Let $\rho(\top) = \{C_1, \ldots, C_n\}$ be a set of refinements of the $\top$ concept. (This set has to be finite, since ρ is finite.) Let m be a natural number larger than the maximum of the *quantifier depths* (depth of the nesting of quantifications) of the concepts in $\rho(\top)$. We construct a concept D as follows:

[2] Our result in fact invalidates a claim made in [4] stating that there can be no complete and finite $\mathcal{ALER}$ refinement operator.

$$D = \underbrace{\forall r. \ldots . \forall r}_{m-\text{times}} . \bot \sqcup \underbrace{\exists r. \ldots . \exists r}_{(m+1)-\text{times}} . \top$$

We have shown, which is the main part of the proof, that there exists no concept with a quantifier depth smaller than m, which strictly subsumes D and is not equivalent to $\top$. This means that $C_1, \ldots, C_n$ do not subsume D (note that the properness of ρ implies that $C_1, \ldots, C_n$ are not equivalent to $\top$), so D cannot be reached by further refinements from any of these concepts. Since $C_1, \ldots, C_n$ are the only refinements of $\top$, it is impossible to reach D from $\top$. Thus ρ is not complete, which is what we wanted to show.

However, if the requirement of finiteness is dropped, corresponding operators exist.

Proposition 5. *For any language $\mathcal{L}$, which satisfies the conditions stated in Definition 3, there exists a complete and proper $\mathcal{L}$ refinement operator.*

Propositions 3, 4, and 5 state that for complete refinement operators, which are usually desirable, one has to sacrifice properness or finiteness. Both combinations can be useful in practise. As noted above, we have developed a complete and finite operator in [11], because the finiteness property was important in this context. However, in [13] we have chosen to develop a complete and proper operator, in an ILP style top-down learning algorithm, because it is easier to overcome the problem of an infinite operator in this context.

We will now look at non-redundancy.

Proposition 6. *For any language $\mathcal{L}$, which satisfies the conditions stated in Definition 3, there exists a complete and non-redundant $\mathcal{L}$ refinement operator.*

Proof. We will prove the result by showing that each complete operator can be transformed to a complete and non-redundant operator. Note that in the following, we will use the role r to create concepts with a certain depth. If N_R does not contain any role, the desired effect can also be achieved by using conjunctions or disjunctions of $\top$ and $\bot$, but this would render the proof less readable.

We will use the fact that the set of concepts in $\mathcal{L}$ is countably infinite. The countability already follows from the fact that there is just a finite number of concepts with a given length. Hence, we can divide the set of all concepts in finite subsets, where each subset contains all concepts of the same length. We can then start enumerating concepts starting with the subset of concepts of length 1, then length 2 etc. Thus, there is a bijective function $f : \mathcal{L} \mapsto \mathbf{N}$, which assigns a different number to each concept $C \in \mathcal{L}$. We denote the inverse function mapping numbers to concepts by f^{inv}.

Now, we modify a given complete refinement operator ρ, e.g. the operator in the proof of Proposition 5 (see [12] for details), in the following way: For any concept C, $\rho(C)$ is modified by changing any element $D \in \rho(C)$ with depth d to

$$D \sqcap \underbrace{\forall r. \ldots \forall r}_{d+1 \text{ times}} . (\underbrace{\top \sqcap \cdots \sqcap \top}_{f(C) \text{ times}})$$

We claim that the resulting operator, which we want to denote by ρ' is complete and non-redundant.

The completeness of ρ' follows from the completeness of ρ, since the construct we have added does not change the meaning (it is equivalent to $\top$).

To prove non-redundancy, we will first define a function ρ^{inv}, which maps conjunctions, which contain at least one element of the form $\forall r. \ldots \forall r.(\top \sqcap \cdots \sqcap \top)$, to concepts:

$$\rho^{\text{inv}}(C \sqcap \underbrace{\forall r. \ldots \forall r.\underbrace{(\top \sqcap \cdots \sqcap \top)}_{n \text{ times}}}_{\text{element with largest depth}}) = f^{\text{inv}}(n)$$

We can see that $D \in \rho'(C)$ implies $\rho^{\text{inv}}(D) = C$, so ρ^{inv} allows to invert a refinement step of ρ'. Furthermore, we have $C \simeq D$ implies $\rho^{\text{inv}}(C) = \rho^{\text{inv}}(D)$, because ρ^{inv} treats all weakly equal concepts in the same way.

By the definition of redundancy, there needs to be a concept C and concepts D_1, D_2 with $D_1 \simeq D_2$, such that there is a refinement chain from C to D_1 and a different refinement chain from C to D_2. However, if D_1 and D_2 are weakly equal, then $\rho^{\text{inv}}(D_1) = \rho^{\text{inv}}(D_1)$, and by continuing to apply ρ^{inv} we will reach C. Hence the two refinement chains from C to D_1 and D_2, respectively, cannot be different. Thus, ρ' is non-redundant.

Essentially, the proof of Proposition 6 is done by showing how to make complete operators non-redundant by using the fact that the set of concepts in any considered language $\mathcal{L}$ is countably infinite. We note, however, that under a mild additional assumption Proposition 6 no longer holds.

Proposition 7. *Let ρ be an arbitrary $\mathcal{L}$ refinement operator, where $\mathcal{L}$ satisfies the conditions stated in Definition 3, and for any concept C, $\rho^*(C)$ contains only finitely many different concepts equivalent to $\bot$. Then ρ is not complete and non-redundant.*

The assumption made in the proposition is indeed mild: If we have $\bot \in \rho^*(C)$ for any concept C, then it is already satisfied. The assumption is made in order to disallow pure theoretical constructions, as in the proof of Proposition 6, which use syntactic concept extensions, that do not alter the semantics of a concept, to ensure non-redundancy. In fact, the result also holds under an even milder, but more technical assumption, see our report [12].

As a consequence, completeness and non-redundancy usually cannot be combined. It is often desirable to have (weakly) complete operators, but in order to have a full analysis of $\mathcal{L}$ refinement operators we will now also investigate incomplete operators.

Proposition 8. *For any language $\mathcal{L}$, which satisfies the conditions stated in Definition 3, there exists a finite, proper, and non-redundant $\mathcal{L}$ refinement operator.*

Proof. The following operator has the desired properties:

$$\rho(C) = \begin{cases} \{\bot\} & \text{if } C \not\equiv \bot \\ \emptyset & \text{otherwise} \end{cases}$$

It is obviously finite, because it maps concepts to sets of cardinality at most 1. It is non-redundant, because it only reaches the bottom concept and there exists no refinement chain of length greater than 2. It is proper, because all concepts, which are not equivalent to the bottom concept strictly subsume the bottom concept.

The corresponding upward operator is:

$$\phi(C) = \begin{cases} \{\top\} & \text{if } C \not\equiv \top \\ \emptyset & \text{otherwise} \end{cases}$$

The arguments for its finiteness, properness, and non-redundancy are analogous to the downward case.

We can now summarise the results we have obtained so far.

Theorem 1. *Let $\mathcal{L}$ be a language, which satisfies the conditions stated in Definition 3. Considering the properties completeness, properness, finiteness, and non-redundancy the following are maximal sets of properties (in the sense that no other of the mentioned properties can be added) of $\mathcal{L}$ refinement operators:*

1. $\{complete, finite\}$
2. $\{complete, proper\}$
3. $\{non\text{-}redundant, finite, proper\}$

All results hold under the mild hypothesis stated in Proposition 7.

A property we have not yet considered is weak completeness. Usually weak completeness is sufficient, because it allows to search for a good concept starting from $\top$ downwards (top-down approach) or from $\bot$ upwards (bottom-up approach).

We will see that we get different results when considering weak completeness instead of completeness.

Proposition 9. *For any language $\mathcal{L}$, which satisfies the conditions stated in Definition 3, there exists a weakly complete, non-redundant, and proper $\mathcal{L}$ refinement operator.*

The result just given also holds when properness is replaced by finiteness.

Proposition 10. *For any language $\mathcal{L}$, which satisfies the conditions stated in Definition 3, there exists a weakly complete, non-redundant, and finite $\mathcal{L}$ refinement operator.*

However, properness and finiteness cannot be achieved at the same time if weak completeness is imposed. To show this, we can use the same proof as for

Proposition 8, where we have shown that for any finite and proper refinement operator, there exists a concept, which cannot be reached from $\top$.

Corollary 1. *For any language $\mathcal{L}$, which satisfies the conditions stated in Definition 3, there exists no weakly complete, finite, and proper $\mathcal{L}$ refinement operator.*

The result of the previous observations is that, when requiring only weak completeness instead of completeness, non-redundant operators are possible.[3]

The following theorem is the result of the full analysis of the desired properties of $\mathcal{L}$ refinement operators.

Theorem 2 (Property Theorem). *Let $\mathcal{L}$ be a language, which satisfies the conditions stated in Definition 3. Considering the properties completeness, weak completeness, properness, finiteness, and non-redundancy the following are maximal sets of properties (in the sense that no other of the mentioned properties can be added) of $\mathcal{L}$ refinement operators:*

1. {*weakly complete*, *complete*, *finite*}
2. {*weakly complete*, *complete*, *proper*}
3. {*weakly complete*, *non-redundant*, *finite*}
4. {*weakly complete*, *non-redundant*, *proper*}
5. {*non-redundant*, *finite*, *proper*}

All results hold under the mild hypothesis stated in Proposition 7.

Remark 1. Instead of using subsumption ($\sqsubseteq$) as an ordering over concepts, we can also use subsumption with respect to a TBox $\mathcal{T}$ ($\sqsubseteq_{\mathcal{T}}$). Theorem 2 will also hold in this case.

The results also hold if we consider equality ($=$) instead of weak equality ($\simeq$) as equivalence relation on concepts.

5 Related Work

Related work can essentially be divided in two parts. The first part is research which is directly connected to learning in description logics. The second part is research about refinement operators in general, often connected with the learning of logic programs. We will describe both in turn.

In [4] a refinement operator for $\mathcal{ALER}$ has been designed to obtain a top-down learning algorithm for this language. Properties of refinement operators in this language were discussed and some claims were made, but a full formal

[3] Reducing completeness to weak completeness to obtain non-redundancy seems to be an interesting option for the development of practical operators. However, this approach is problematic since it is not immediately clear how this idea could be put into practise. For instance, a weakly complete and non-redundant downward refinement operator ρ cannot allow refinements of the form $C \rightsquigarrow C \sqcap \top$ or $C \rightsquigarrow C \sqcap \rho(\top)$: A weakly complete operator, which allows one of these refinement steps is also complete and is therefore usually redundant by Proposition 7.

analysis was not performed. Our work generalises the results in this article, refutes them in one case, investigates more property combinations, and proves each claim. In [6,9] learning algorithms for description logics, in particular for the language $\mathcal{ALC}$ were created, which also make use of refinement operators. Instead of using the classical approach of combining refinement operators with a search heuristic, they developed an example driven learning method. [6] stated that an investigation of the properties of refinement operators in description logics, as we have done in this article, is required. In [7] downward refinement for $\mathcal{ALN}$ was analysed using a clausal representation of DL concepts. This article also states that further investigation of the properties of refinement operators in description logics is required. Refinement operators have also been dealt with in hybrid systems. In [14] ideal refinement for learning $\mathcal{AL}$-log, a language that merges DATALOG and $\mathcal{ALC}$, was investigated. Based on the notion of $\mathcal{B}$-subsumption, an ideal refinement operator was created. In [5,10] learning algorithms for description logics without refinement operators were analysed.

In the area of Inductive Logic Programming [18] considerable efforts have been made to analyse the properties of refinement operators. Note, that in general using refinement operators for clauses to learn in description logics is possible, but usually not a good choice as shown in [4]. However, the theoretical foundations of refinement operators also apply to description logics, which is why we want to mention work in this area here.

A milestone in Machine Learning [15] in general was the Model Inference System in [19]. Shapiro describes how refinement operators can be used to adapt a hypothesis to a sequence of examples. Afterwards, refinement operators became widely used as (part of) a learning method. [20] have found some general properties of refinement operators in quasi-ordered spaces. Nonexistence conditions for ideal refinement operators relating to infinite ascending and descending refinement chains and covers have been developed. This has been used earlier to show that ideal refinement operators for clauses ordered by θ-subsumption do not exist [20]. Unfortunately, we could not make use of these results, because proving properties of covers in description logics without the restriction to a specific language is likely to be harder than directly proving the results.

[17] discussed refinement for different versions of subsumption, in particular weakenings of logical implication. In [16] it was shown how to extend refinement operators to learn general prenex conjunctive normal form. Perfect, i.e. weakly complete, locally finite, non-redundant, and minimal operators, were discussed in [2]. Since such operators do not exist for clauses ordered by θ-subsumption [20], weaker versions of subsumption were considered. This was later extended to theories, i.e. sets of clauses [8]. A less widely used property of refinement operators, called flexibility, was discussed in [3]. Flexibility essentially means that previous refinements of an operator can influence the choice of the next refinement. The article discusses how flexibility interacts with other properties and how it influences the search process in a learning algorithm.

6 Conclusions

We have presented a comprehensive analysis of properties of refinement operators. The results are summarised in Theorems 1 and 2. In particular, we have shown that ideal refinement operators for description logics cannot exist. We have also shown in detail which combinations of properties are in general achievable. This analysis is fundamental for using refinement operators in description logics and was requested in [6,7].

After the derivation of these results, two learning systems have been developed, which use the Property Theorem as theoretical foundation. They are reported on elsewhere [11,13]. Both systems are fully implemented and have shown promising results in evaluations.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
2. Badea, L., Stanciu, M.: Refinement operators can be (weakly) perfect. In: Džeroski, S., Flach, P.A. (eds.) ILP 1999. LNCS (LNAI), vol. 1634, pp. 21–32. Springer, Heidelberg (1999)
3. Badea, L.: Perfect refinement operators can be flexible. In: Horn, W. (ed.) Proceedings of the 14th European Conference on Artificial Intelligence, August 2000, pp. 266–270. IOS Press, Amsterdam (2000)
4. Badea, L., Nienhuys-Cheng, S.-H.: A refinement operator for description logics. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 40–59. Springer, Heidelberg (2000)
5. Cohen, W.W., Hirsh, H.: Learning the classic description logic: Theoretical and experimental results. In: Doyle, P.T.J., Sandewall, E. (eds.) Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning, Bonn, FRG, May 1994, pp. 121–133. Morgan Kaufmann, San Francisco (1994)
6. Esposito, F., Fanizzi, N., Iannone, L., Palmisano, I., Semeraro, G.: Knowledge-intensive induction of terminologies from metadata. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 441–455. Springer, Heidelberg (2004)
7. Fanizzi, N., Ferilli, S., Iannone, L., Palmisano, I., Semeraro, G.: Downward refinement in the ALN description logic. In: 4th International Conference on Hybrid Intelligent Systems (HIS 2004), Kitakyushu, Japan, December 2004, pp. 68–73. IEEE Computer Society, Los Alamitos (2004)
8. Fanizzi, N., Ferilli, S., Di Mauro, N., Basile, T.M.A.: Spaces of theories with ideal refinement operators. In: Gottlob, G., Walsh, T. (eds.) IJCAI 2003, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003, pp. 527–532. Morgan Kaufmann, San Francisco (2003)
9. Iannone, L., Palmisano, I.: An algorithm based on counterfactuals for concept learning in the semantic web. In: Ali, M., Esposito, F. (eds.) Innovations in Applied Artificial Intelligence. Proceedings of the 18th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Bari, Italy, June 2005, pp. 370–379 (2005)

10. Kietz, J.-U., Morik, K.: A polynomial approach to the constructive induction of structural knowledge. Machine Learning 14, 193–217 (1994)
11. Lehmann, J.: Hybrid learning of ontology classes. In: Perner, P. (ed.) MLDM 2007. LNCS (LNAI), vol. 4571, Springer, Heidelberg (2007)
12. Lehmann, J., Hitzler, P.: Foundations of refinement operators for description logics. In: Technical report. University of Leipzig (2007), http://www.jens-lehmann.org
13. Lehmann, J., Hitzler, P.: Foundations of refinement operators for description logics. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS (LNAI), vol. 4894, pp. 161–174. Springer, Heidelberg (2007)
14. Lisi, F.A., Malerba, D.: Ideal refinement of descriptions in AL-log. In: Horváth, T., Yamamoto, A. (eds.) ILP 2003. LNCS (LNAI), vol. 2835, pp. 215–232. Springer, Heidelberg (2003)
15. Mitchell, T.: Machine Learning. McGraw Hill, New York (1997)
16. Nienhuys-Cheng, S.-H., Van Laer, W., Ramon, J., De Raedt, L.: Generalizing refinement operators to learn prenex conjunctive normal forms. In: Džeroski, S., Flach, P.A. (eds.) ILP 1999. LNCS (LNAI), vol. 1634, pp. 245–256. Springer, Heidelberg (1999)
17. Nienhuys-Cheng, S.H., van der Laag, P.R.J., van der Torre, L.W.N.: Constructing refinement operators by decomposing logical implication. In: Torasso, P. (ed.) AI*IA 1993. LNCS, vol. 728, pp. 178–189. Springer, Heidelberg (1993)
18. Nienhuys-Cheng, S.-H., de Wolf, R. (eds.): Foundations of Inductive Logic Programming. LNCS, vol. 1228. Springer, Heidelberg (1997)
19. Shapiro, E.Y.: Inductive inference of theories from facts. In: Lassez, J.L., Plotkin, G.D. (eds.) Computational Logic: Essays in Honor of Alan Robinson, pp. 199–255. MIT Press, Cambridge (1991)
20. van der Laag, P.R.J., Nienhuys-Cheng, S.-H.: Existence and nonexistence of complete refinement operators. In: Bergadano, F., De Raedt, L. (eds.) ECML 1994. LNCS, vol. 784, pp. 307–322. Springer, Heidelberg (1994)

A Relational Hierarchical Model for Decision-Theoretic Assistance

Sriraam Natarajan, Prasad Tadepalli, and Alan Fern

School of EECS,
Oregon State University, Corvallis, Oregon, USA

Abstract. Building intelligent assistants has been a long-cherished goal of AI and many were built and fine-tuned to specific application domains. In recent work, a domain-independent decision-theoretic model of assistance was proposed, where the task is to infer the user's goal and take actions that minimize the expected cost of the user's policy. In this paper, we extend this work to domains where the user's policies have rich relational and hierarchical structure. Our results indicate that relational hierarchies allow succinct encoding of prior knowledge for the assistant, which in turn enables the assistant to start helping the user after a relatively small amount of experience.

1 Introduction

There has been a growing interest in developing intelligent assistant systems that help users in a variety of tasks ranging from washing hands to travel planning [2,6,3]. The emphasis in these systems has been to provide a well-engineered domain-specific solution to the problem of reducing the users' cognitive load in their daily tasks. A decision-theoretic model was proposed recently to formalize the general problem of assistantship as a partially observable Markov decision process (POMDP). In this framework, the assistant and the user interact in the environment to change its state. The goal of the assistant is to take actions that minimize the expected cost of completing the user's task [9]. In most situations, however, the user's task or goal[1] is not directly observable to the assistant, which makes the problem of quickly inferring the user's goals from observed actions critically important. One approach to goal inference [9] is to learn a probabilistic model of the user's policy for achieving various goals and then to compute a posterior distribution over goals given the current observation history. However, for this approach to be useful in practice, it is important that the policy be learned as early in the lifetime of the assistant as possible. We call this the problem of "early assistance", which is the main motivation behind this work.

One solution to the early assistance problem, advocated in [9], is to assume that (a) the user's policy is optimal with respect to their goals and actions, the so called "rationality assumption," and that (b) the optimal policy can be computed quickly by knowing the goals, the "tractability assumption." Under

[1] In this work, we use the words task and goal interchangeably.

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 175–190, 2008.

these assumptions, the user's policy for each goal can be approximated by an optimal policy, which may be quickly computed. Unfortunately in many real world domains, neither of these assumptions is realistic. Real world domains are too complex to allow tractable optimal solutions. The limited computational power of the user renders the policies to be locally optimal at best.

In this paper, we propose a different solution to the early assistance problem, namely constraining the user's policies using prior domain knowledge in the form of hierarchical and relational constraints. Consider an example of a desktop assistant similar to CALO [4] that helps an academic researcher. The researcher could have some high level tasks like writing a proposal, which may be divided into several subtasks such as preparing the cover page, writing the project description, preparing the budget, completing the biography, etc. with some ordering relationships between them. We expect that an assistant that knows about this high level structure would better help the user. For example, if the budget cannot be prepared before the cover page is done, the assistant would not consider that possibility and can determine the user's task faster. In addition to the hierarchical structure, the tasks, subtasks, and states have a class and relational structure. For example, the urgency of a proposal depends on the closeness of the deadline. The deadline of the proposal is typically mentioned on the web page of the agency to which the proposal is addressed. The collaboration potential of an individual on a proposal depends on their expertise in the areas related to the topic of the proposal. Knowing these relationships and how they influence each other could make the assistant more effective.

This work extends the assistantship model to hierarchical and relational settings, building on the work in hierarchical reinforcement learning[10] and statistical relational learning (SRL).We extend the assistantship framework of [9] by including parameterized task hierarchies and conditional relational influences as prior knowledge of the assistant. We compile this knowledge into an underlying Dynamic Bayesian network and use Bayesian network inference algorithms to infer the distribution of user's goals given a sequence of their atomic actions. We estimate the parameters for the user's policy and influence relationships by observing the users' actions. Once the user's goal distribution is inferred, we determine an approximately optimal action by estimating the Q-values of different actions using rollouts and picking the action that has the least expected cost.

We evaluate our relational hierarchical assistantship model in two different toy domains and compare it to a propositional flat model, propositional hierarchical model, and a relational flat model. Through simulations, we show that when the prior knowledge of the assistant matches the true behavior of the user, the relational hierarchical model provides superior assistance in terms of performing useful actions. The relational flat model and the propositional hierarchical model provide better assistance than the propositional flat model, but fall short of the performance of the relational hierarchical approach.

The rest of the paper is organized as follows: Section 2 summarizes the basic decision-theoretic assistance framework, which is followed by the relational

hierarchical extension in Section 3. Section 4 presents the experiments and results, Section 5 outlines some related work and Section 6 concludes the paper.

2 Decision-Theoretic Assistance

In this section, we briefly describe the decision-theoretic model of assistance of [9] which forms the basis of our work. In this setting, there is a user acting in the environment and an assistant that observes the user and attempts to assist him. The environment is modeled as an MDP described by the tuple $\langle W, A, A', T, C, I \rangle$, where W is a finite set of world states, A is a finite set of user actions, A' is a finite set of assistant actions, and $T(w, a, w')$ is a transition function that represents the probability of transitioning to state w' given that action $a \in A \cup A'$ is taken in state w. C is an action-cost function that maps $W \times (A \cup A')$ to real numbers, and I is an initial state distribution over W. An episodic setting is assumed, where the user chooses a goal and tries to achieve it. The assistant observes the user's actions and the world states but not the goal. After every user's action, the assistant gets a chance to take one or more actions ending with a **noop** action, after which the user gets a turn. The objective is to minimize the sum of the costs of user and assistant actions.

The user is modeled as a stochastic policy $\pi(a|w, g)$ that gives the probability of selecting action $a \in A$ given that the user has goal g and is in state w. The objective is to select an assistant policy π' that minimizes the expected cost given the observed history of the user. The environment is only partially observable to the assistant since it cannot observe the user's goal. It can be modeled as a POMDP, where the user is treated as part of the environment.

In [9], the assistant POMDP is solved approximately, by first estimating the goal of the user given the history of his actions, and then selecting the best assistive action given the posterior goal distribution. One of the key problems in effective assistantship is to learn the task quickly enough to start helping the user as early as possible. In [9], this problem is solved by assuming that the user is rational, i.e., he takes actions to minimize the expected cost. Further, the user MDP is assumed to be tractably solvable for each goal. Hence, their system solves the user MDP for each goal and uses it to initialize the user's policy.

Unfortunately the dual assumptions of tractability MDP and rationality make this approach too restrictive to be useful in real-world domains that are too complicated for any user to approach perfect rationality. We propose a knowledge-based approach to the effective assistantship problem that bypasses the above two assumptions. We provide the assistant with partial knowledge of the user's policy, in the form of a task hierarchy with relational constraints on the subtasks and their parameters. Given this strong prior knowledge, the assistant is able to learn the user's policy quickly by observing his actions and updating the policy parameters. We appropriately adopt the goal estimation and action selection steps of [9] to the new structured policy of the user and show that it performs significantly better than the unstructured approach.

3 A Relational Hierarchical Model of Assistance

In this section, we propose a relational hierarchical representation of the user's policy and show its use for goal estimation and action selection.

3.1 Relational Hierarchical Policies

Users in general, solve difficult problems by decomposing them into a set of smaller ones with some ordering constraints between them. For example, proposal writing might involve writing the project description, preparing the budget, and then getting signatures from proper authorities. Also, the tasks have a natural class-subclass hierarchy, e.g., submitting a paper to ICML and IJCAI might involve similar parameterized subtasks. In the real world, the tasks are chosen based on some attributes of the environment or the user. For instance, the paper the user works on next is influenced by the closeness of the deadline. It is these kinds of relationships that we want to express as prior knowledge so that the assistant can quickly learn the relevant parameters of the policy. We model the user as a stochastic policy $\pi(a|w, T, O)$ that gives the probability of selecting action $a \in A$ given that the user has goal stack T and is in state w. O is the history of the observed states and actions. Learning a flat, propositional representation of the user policy is not practical in many domains. Rather, in this work, we represent the user policy as a *relational task hierarchy* and speed up the learning of the hierarchy parameters via the use of *conditional influence statements* that constrain the space of probabilistic dependencies.

Relational Task Hierarchies. A relational task hierarchy is specified over a set of variables, domain constants, and predicate symbols. There are predicate symbols for representing properties of world states and specifying task names. The task predicates are divided into primitive and abstract tasks. Primitive task predicates will be used to specify ground actions in the MDP that can be directly executed by the user. Abstract task predicates will be used to specify non-primitive procedures (that involve calling subtasks) for achieving high-level goals. Below we will use the term *task stack* to mean a sequence of ground task names (i.e. task predicates applied to constants).

A relational task hierarchy will be composed of relational task schemas which we now define.

Definition 1 (Relational Task Schema). *A relational task schema is either: 1) A primitive task predicate applied to the appropriate number of variables, or 2) A tuple $\langle N, S, R, G, P\rangle$, where the task name N is an abstract task predicate applied to a set of variables V, S is a set of child relational task schemas (i.e. the subtasks), R is a set of logical rules over state, task, and background predicates that are used to derive a candidate set of ground child tasks in a given situation, G is a set of rules that define the goal conditions for the task, and $P(s|T, w, O)$ is a probability distribution that gives the probability of a ground child task s conditioned on a task stack T, a world state w, and an observation history O.*

Each way of instantiating the variables of a task schema with domain constants yields a ground task. The semantics of a relational task schema specify what it means for the user to "execute to completion" a particular ground task as follows. As the base case, a primitive ground task is executed-to-completion by simply executing the corresponding primitive MDP action until it terminates, resulting in an updated world state.

An abstract ground task, can intuitively be viewed as specifying a stochastic policy over its child subtasks which is executed until its goal condition is satisfied. More precisely, an abstract ground task t is executed-to-completion by repeatedly selecting ground child tasks that are executed-to-completion until the goal condition G is satisfied. At each step given the current state w, observation history O, task stack T, and set of variable bindings B (that include the bindings for t) a child task is selected as follows: 1) Subject to the variable bindings, the rules R are used to derive a set of candidate ground child tasks. 2) From this set we draw a ground task s according to P, properly normalized to only take into account the set of available subtasks. 3) The drawn ground task is then executed-to-completion in the context of variables bindings B' that include the bindings in B along with those in s and a task stack corresponding to pushing t onto T.

Based on the above description, the set of rules R can be viewed as specifying hard constraints on the legal subtasks with P selecting among those tasks that satisfy the constraints. The hard constraints imposed by R can be used restrict the argument of the child task to be of a certain type or may place mutual constraints on variables of the child tasks. For example, we could specify rules that say that the document to be attached in an email should belong to the project that the user is working on. Also, the rules can specify the ordering constraint between the child tasks. For instance, it would be possible to say that to submit a paper the task of writing the paper must be completed first.

We can now define a relational task hierarchy.

Definition 2 (Relational Task Hierarchy). *A relational task hierarchy is rooted acyclic graph whose nodes are relational task schemas that satisfy the following constraints: 1) The root is a special subtask called ROOT. 2) The leaves of the graph are primitive task schemas. 3) There is an arc from node n_1 to node n_2 if and only if the task schema of n_2 is a child of task schema n_1.*

We will use relational task hierarchies to specify the policy of a user. Specifically, the user's actions are assumed to be generated by executing the ROOT task of the hierarchy with an initially empty goal stack and set of variable bindings.

An example of a *Relational Task Hierarchy* is presented in the Figure 1 for a game involving resource gathering and tactical battles. For each task schema we depict some of the variable binding constraints enforced by the R as a logical expression. For clarity we do not depict the ordering constraints imposed by R. From the ROOT task the user has two distinct choices to either gathering a resource, *Gather(R)* or attacking an enemy, *Attack(E)*. Each of these tasks can be achieved by executing either a primitive action (represented with ovals in the figure) or another subtask. For example, to gather a resource, the user needs

to collect the resource (denoted by *Collect(R)*) and deposit the resource at the storage (denoted by *Deposit(R,S)*, which indicates that R is to be deposited in S). Resources are stored in the storages of the same type (for example, gold in a bank, food in a granary etc.), which is expressed as the constraint $R.type = S.type$ in the figure. Once the user chooses to gather a resource (say $gold1$), the value of R in all the nodes that are lower than the node $Gather(R)$ is set to the value $gold1$. R is freed after *Gather* is completed.

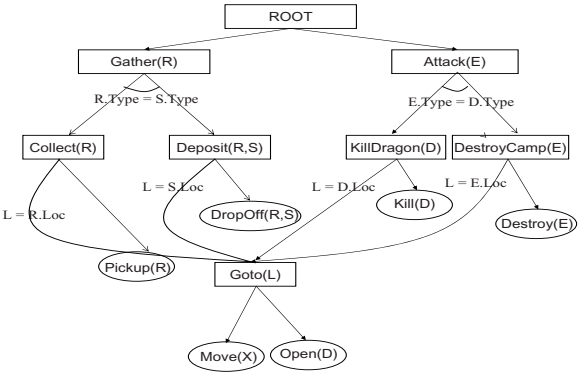


Fig. 1. Example of a task hierarchy of the user. The inner nodes indicate subtasks while the leaves are the primitive actions. The tasks are parameterized and the tasks at the higher level will call the tasks at the lower level.

Conditional Influences: Often it is relatively easy to hand-code the rule sets R that encode hard-constraints on child tasks. It is more difficult to precisely specify the probability distributions for each task schema. In this work, we take the approach of hand-coding a set of conditional influence statements that are used to constrain and hence speedup the learning of these probability distributions. The conditional influences describe the objects and their attributes that influence a subtask choice based on some condition, i.e., these statements serve to capture a distribution over the subtasks given some attributes of the environment ($P(subtask \mid worldstate)$). For example, since there could be multiple storage locations for a resource, the choice of a storage may be influenced by its distance to the resource. While this knowledge can be easily expressed in most SRL formalisms such as Probabilistic Relational Language [18] and Bayesian Logic Programs [15], we give an example in First-Order Conditional Influence Language (FOCIL) [19].

```
If {Goal(Gather(R)),Completed(Collect(R)),Equal(Type(R),Type(S))} then
Distance(Loc(R), Loc(S))) Qinf subgoal(Deposit(R,S))
```

A FOCIL statement of the form $If\{Z(\alpha)\}$ *then* $Y_1(\alpha), \ldots, Y_k(\alpha)$ $Qinf$ $X(\alpha)$ means that $Y_1(\alpha), \ldots, Y_k(\alpha)$ influence $X(\alpha)$ when $Z(\alpha)$ is true, where α is

a set of logical variables. The above statement captures the knowledge that if R is a resource that has been collected, and S is a storage where R can be stored, the choice of the value of S is influenced by the distance between R and S. The probability of choosing a subtask in a given state is determined solely by the attribute values of the objects mentioned in the conditional influence statement, which puts a strong constraint on the user's policy and makes it easier to learn.

The high level algorithm is presented in table 1. The parameters are updated at the end of the episode using MLE estimates. When an episode is completed, the set of completed tasks and the action trajectories are used to update the parameters of the nodes at different levels.

Table 1. Highlevel algorithm for assistance

- Iitialize DBNs as in Figure 2 incorporating all hard constraints into the CPTs
- For each episode
  - For each time step
    - Observe any task completed
    - Update the posterior distribution of goal stack based on the observation, the hard constraints, and FOCI statements
    - Observe the next action
    - Update the posterior distribution over the tasks in the task stack
    - Compute the best assistive action
  - Update the DBN parameters

3.2 Goal Estimation

In this section, we describe our goal estimation method, given the kind of prior knowledge described in the previous section, and the observations, which consist of the user's primitive actions. Note that the probability of the user's action choice depends in general on not only the pending subgoals, but also on some of the completed subgoals including their variable bindings. Hence, in general, the assistant POMDP must maintain a belief state distribution over the pending and completed subgoals. which we call the "goal structure."

We now define the assistant POMDP. The **state space** is $W \times \mathbf{T}$ where W is the set of world states and $\mathbf{T}$ is the user's goal structure. Correspondingly, the **transition probabilities** are functions between $(w, \mathbf{t})$ and $(w', \mathbf{t})$. Similarly, the **cost** is a function of $\langle state,\ action \rangle$ pairs. The **observation** space now includes the user's actions and their parameters (for example, the resource that is collected, the enemy type that is killed etc).

In this work, we make a simplifying assumption that there is no uncertainty about the completed subtasks. This assumption is justified in our domains, where the completion of each subtask is accompanied with an observation that identifies the subtask that has just completed. This would simplify the inference process as

we do not need to maintain a distribution over the (possibly) completed subtasks. For estimating the user's goal stack, we use a DBN similar to the one used in [16] and present it in Figure 2. T_j^i refers to the task at time-step j and level i in the DAG. O^i refers to the completed subtask at level i. F_j^i is an indicator variable that represents whether T_j^i has been completed and acts as a multiplexer node. If the lower level task is completed and the current task is not completed, the transition function for the current task would reflect choosing an action for the current subtask. If the lower level task is not completed, the current task stays at its current state. If the current task is completed, the value is chosen using a prior distribution over the current task given the higher level tasks.

In the experiments reported in the next section, we compiled the FOCIL statements into a DBN structure by hand. The number of levels of the tasks in the DBN corresponds to the depth of the directed graph in the relational task hierarchy. The values of the different task level nodes will be the instantiated tasks in the hierarchy. For instance, the variable T_j^1 takes values corresponding to all possible instantiations of the second-level tasks. Once the set of possible values for each current task variable in the task is determined, the constraints are used to construct the CPT. For example, the constraint $R.Type = S.Type$ in the Figure 1 implies that a resource of one type can be stored in the storage of the same type. Assume that the user is gathering *gold*. Then in the CPT corresponding to $P(T_j^2 = Store(S, gold) \mid T_j^1 = Gather(gold)$, all the entries except the ones that correspond to a bank are set to 0. The rules R of the task schema determine the non-zero entries of the CPTs, while the FOCIL statements constrain the distributions further. Note that, in general, the subtasks completed at a particular level influence the distribution over the current subtasks at the same level through the hard constraints, which include ordering relationships. In our experiments, however, we have chosen to not explicitly store the completed subtasks at any stage since the ordering of subtasks has a special structure. The subtasks are partitioned into small unordered groups, where the groups are totally ordered. This allows us to maintain a small memory of only the completed subtasks in the current group.

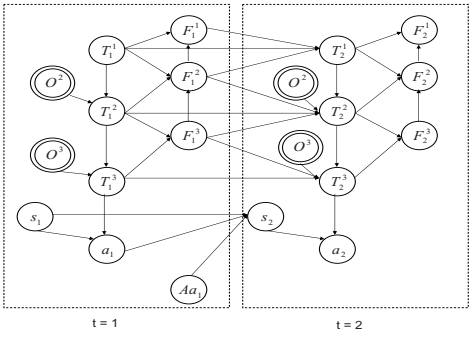


Fig. 2. Dynamic Bayesian network that is used to infer the user's goal

To illustrate the construction of the DBN given the hierarchy and influence statements better, let us consider the example presented in Figure 1. Assume that the user chooses to gather $g1$ (i.e., gold from location 1). Once the episode begins, the variables in the DBN are instantiated to the corresponding values. The task at the highest level T_j^1, would take values from the set $\langle\, Gather(g1), Gather(g2), Gather(w1), Gather(w2), Destroy(e1), Destroy(e2) \,\rangle$, assuming that there are 2 gold and wood locations and 2 enemies. Similarly, the tasks at level n of the DBN would assume values corresponding to the instantiation of the nodes at the n^{th} level of the hierarchy. The conditional influence statements are used to obtain a prior distribution over the goal stack only after every subtask is finished or to minimize uncertainty and retain tractability. Once the prior is obtained, the posterior over the goal stack is updated after every user action. For example, once the user finishes the subtask of $collect(g1)$, the relational structure would restrict the set of subgoals to depositing the resource and the conditional influence statements would provide a prior over the storage locations. Once the highest level task of *Gather* is completed, the DBN parameters are updated using the complete set of observations. Our hypothesis that we verify empirically is that, the relational structure and the conditional influence statements together provide a strong prior over the task stack which enables fast learning.

Given this DBN, we need to infer the value of $P(T_j^{1:d} \mid T_{j-1}^{1:d}, F_{j-1}^{1:d}, a_j, O^{1:d})$, where d is the depth of the DAG i.e, infer the posterior distribution over the user's goal stack given the observations (the user actions in our case) and the completed goal stack. As we have mentioned, we are not considering the completed subgoals due to the fact that most of our constraints are total order and there is no necessity of maintaining them. Since we always estimate the current goal stack given the current action and state, we can approximate the DBN inference as a BN inference for the current time-step. The other issue is the learning of parameters of the DBN. At the end of every episode, the assistant updates the parameters of the DBN based on the observations in that episode using maximum likelihood estimates with Laplace correction. Since the model is inherently relational, we can exploit parameter tying between similar objects and hence accelerate the learning of parameters. The parameter learning in the case of relational models is significantly faster as demonstrated by our experiments.

It should be noted that Fern et.al solved the user MDP and used the values to initialize the priors for the user's action models. Though it seems justifiable, it is not always possible to solve the user MDP. We show in our experiments that even if we begin with an uniform prior for the action models, the relations and the hierarchical structure would enable the assistant to be useful even in the early episodes.

3.3 Action Selection

Given the assistant POMDP M and the distribution over the user's goal stack $P(T^{1:d} \mid O_j)$, where O_j are the observations, we can compute the value of assistive actions. Following the approach of [9], we approximate the assistant POMDP

with a series of MDPs $M(t^{1:d})$, for each possible goal stack $t^{1:d}$. Thus, the heuristic value of an action a in a world state w given the observations O_j at time-step j would now correspond to,

$$H(w, a, O_j) = \sum_{t^{1:d}} Q_{t^{1:d}}(w, a) \cdot P(t^{1:d}|O_j)$$

where $Q_{t^{1:d}}(w, a)$ is the value of performing the action a in state w in the MDP $M(t^{1:d})$ and $P(t^{1:d}|O_j)$ is the posterior probability of the goal stack given the observations. Instead of sampling over the goals, we sample over the possible goal stack values. The relations between the different goals would restrict the number of goal-subgoal combinations. If the hierarchy is designed so that the subgoals are not shared between higher level goals, we can greatly reduce the number of possible combinations and hence making the sampling process practically feasible. We verify this empirically in our experiments. To compute the value of $Q_{t^{1:d}}(w, a)$, we use the policy rollout technique [5] where the assumption is that the assistant would perform only one action and assumes that the agent takes over from there and estimates the value by rolling out the user policy. Since the assistant has access to the hierarchy, it chooses the actions subjected to the constraints specified by the hierarchy.

4 Experiments and Results

In this section, we briefly explain the results of simulation of a user in two domains[2]: a gridworld doorman domain where the assistant has to open the right doors to the user's destination and a kitchen domain where the assistant helps the user in preparing food. We simulate a user in these domains and compare different versions of the decision theoretic model and present the results of the comparison. The different models that we compare are: the relational hierarchical model that we presented, a hierarchical model where the goal structure is hierarchical, a relational model where there are objects and relations but there is a flat goal structure and a flat model which is a very naive model with a flat goal structure and no notion of objects are relationships. Our hypothesis is that the relational models would benefit from parameter tying and hence can learn the parameters faster and would offer better assistance than their propositional counterparts at earlier episodes. Similarly, the hierarchical model would make it possible to decompose the goal structure thus making it possible to learn faster. We demonstrate through experiments that the combination of relational and hierarchical models would enable the assistant to be more effective than the assistant that uses either of these models.

4.1 Doorman Domain

In this domain, the user is in a gridworld where each grid cell has 4 doors that the user has to open to navigate to the adjacent cell (see Figure 3.a). The hierarchy

[2] These are modification to the domains presented by Fern et.al [9].

presented in Figure 1.*a* was used as the user's goal structure. The goals of the user are to *Gather* a resource or to *Attack* an enemy. To *gather* a resource, the user has to *collect* the resource and *deposit* it at the corresponding location. Similarly, to *destroy* an enemy, the user has to kill the *dragon* and *destroy* the castle. There are different kinds of resources, namely *food* and *gold*. Each resource can be stored only in a storage of its own type (i.e, *food* is stored in *granary* and *gold* is stored in *bank*). There are 2 locations for each of the resources and its storage. Similarly there are 2 kinds of enemy *red* and *blue*. The user has to kill the *dragon* of a particular kind and *destroy* the castle of the same kind. The episode ends when the user achieves the highest level goal. The actions that the user can perform are to move in 4 directions, open the 4 doors, pick up, put down and attack. The assistant can only open the doors or perform a *noop*. The door closes after one time-step so that at any time only one door is open. The goal of the assistant is to minimize the number of doors that the user needs to open. The user and assistant take actions alternately in this domain. We employed

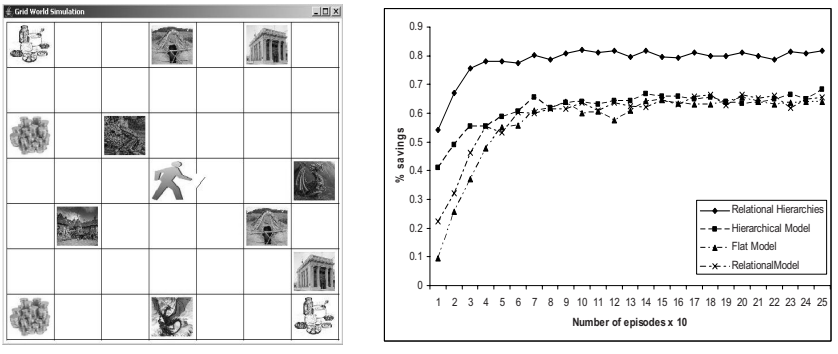


Fig. 3. (a)Doorman Domain. Each cell has 4 doors that the user has to open to navigate to the adjacent cell. The goal of the assistant is to minimize the number of doors that the user has to open. (b)Learning curves for the 4 algorithms in the doorman domain. The y-axis presents the average savings for the user due to the assistant.

four versions of the assistant that models the user's goal structure: one that models the structure as a relational hierarchical model, second which assumes a hierarchical goal structure but no relational structure (i.e., the model does not know that the 2 gold locations are of the same type etc and thus cannot exploit parameter tying), third which assumes a relational structure of user's goal but assumes flat goals and hence does not know the relationship between collect and deposit of subtasks, and the fourth that assumes a flat goal structure. A state is a tuple $\langle s, d \rangle$, where s stands for the the agent's cell and d is the door that is open. For the two flat cases, there is a necessity include variables such as *carry* that can take 5 possible values and *kill* that take 3 values to capture the state

of the user having collected a resource or killed the dragon before reaching the eventual destination. Hence the state space of the 2 flat models is 15 times more than that of the hierarchical one.

To compare the 4 algorithms, we solved the underlying hierarchical MDP and then used the Q-values to simulate the user. For each episode, the higher level goals are chosen at random and the user attempts to achieve the goal. We calculate usefulness of the assistant as the ratio of the correct doors that it opens to the total number of doors that are needed to be opened for the user to reach his goal which is a worst-case measure of the cost savings of the user. We average the usefulness every 10 episodes. The user's policy is hidden from the assistant in all the algorithms and the assistant learns the user policy as and when the user performs his actions. The relational model captures the relationship between the resources and storage and between the dragon's type and the castle's type. The hierarchical model captures the relationship between the different goals and subgoals, for instance, that the user has to collect some resource in order to deposit it, etc. The hierarchical relational model has access to both the kinds of knowledge and also to the knowledge that the distance to the storage location influences the choice of the storage location.

The results are presented in Figure 3.*b*. The graph presents the average usefulness of the assistant after every 10 episodes. As can be seen from the figure, the relational hierarchical assistant is more useful than the other models. In particular, it can exploit the prior knowledge effectively as demonstrated by the rapid increase in the usefulness in earlier episodes. The hierarchical and relational models also exploit the prior knowledge and hence have a quicker learning rate than the flat model (as can be seen from the first few episodes of the figure). The hierarchical relational model outperforms the hierarchical model as it can share parameters and hence has to learn a smaller number of parameters. It outperforms the relational model as it can exploit the knowledge of the user's goal structure effectively and can learn quickly at the early stages of an episode.required for computing the best action of the assistant for all the four algorithms. This clearly demonstrates that the hierarchical relational model can be more effective without increasing the computational cost.

4.2 Kitchen Domain

The other experimental domain is a kitchen domain where the user has to cook some dishes. In this domain, the user has 2 kinds of higher-level goals: one in which he could prepare a recipe which contains a main dish and a side dish and the second in which, he could use some instant food to prepare a main dish and a side dish. There are 2 kinds of main dishes and 2 kinds of side dishes that he could prepare from the recipe. Similarly, there are 2 kinds of main dishes and 2 kinds of side dishes that he could prepare from instant food. The hierarchy is presented in Figure 4.*a*. The symbol $\in$ is used to capture the information that the object is part of the plan. For instance, the expression $I \in M.Ing$ means that the parameter to be passed is the ingredient that is used to cook the main dish. The plans are partially ordered. There are 2 shelves with 3 ingredients each. The

shelves have doors that must be opened before fetching ingredients and only one door can be open at a time.

The state consists of the contents of the bowl, the ingredient on the table, the mixing state and temperature state of the ingredient (if it is in the bowl) and the door that is open. The user's actions are: open the doors, fetch the ingredients, pour them into the bowl, mix, heat and bake the contents of the bowl, or replace an ingredient back to the shelf. The assistant can perform all user actions except for pouring the ingredients or replacing an ingredient back to the shelf. The cost of all non-pour actions is -1. Unlike in the doorman domain, here it is not necessary for the assistant to wait at every alternative time step. The assistant continues to act until the **noop** becomes the best action according to the heuristic. The episode begins with all the ingredients in the shelf and the doors closed. The episode ends when the user achieves the goal of preparing a main dish and a side dish either with the recipe or using instant food.

The savings is the ratio of the correct non-pour actions that the assistant has performed to the number of actions required for the goal. Similar to the other domain, we compared 4 different types of models of assistance. The first is the hierarchical relational model that has the knowledge of the goal-subgoal hierarchy and also has the relationship between the subgoals themselves. It knows that the type of the main dish influences the choice of the side dish. The second model is the hierarchical model, that has the notions of the goals and subgoals but no knowledge of the relationship between the main dishes and the side dishes and thus has more number of parameters to learn. The relational model assumes that there are two kinds of food namely the one prepared from recipe and one from instant food and does not possess any knowledge about the hierarchical goal structure. The flat model considers the preparation of each of the 8 dishes as a separate goal and assists the user. Both the flat model and the relational model assume that the user is always going to prepare the dishes in pairs but do not have the notion of main dish and side dishes or the ordering constraints between them.

The results are presented in Figure 4.*b*. As can be seen, the hierarchical models greatly dominate the flat ones. Among the models, the relational models have a

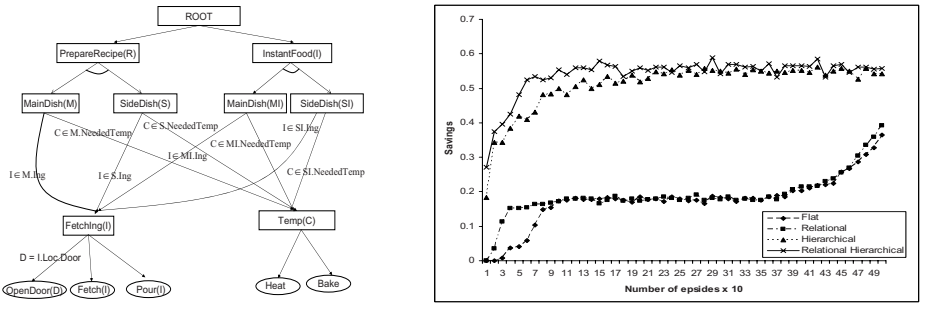


Fig. 4. (a)The kitchen domain hierarchy. (b)Learning curves of the different algorithms.

faster learning rate than their propositional counterparts. They perform better in the earlier few episodes which clearly demonstrates that relational background knowledge accelerates learning. In this domain, the hierarchical knowledge seems to dominate the relational knowledge. This is due to the fact that all the subgoals are similar (i.e, each of them is preparing some kind of food) and the hierarchical knowledge clearly states the ordering of these subgoals. The relational hierarchical model has a better savings rate in the first few episodes as it has a fewer parameters to learn. Both the flat model and the relational model eventually converged on the same *savings* after 700 episodes. These results demonstrate that though all the models can eventually converge to the same value, the relational hierarchical model converges in early episodes.

5 Related Work

Most of the decision-theoretic assistants have been formulated as POMDPs that are approximately solved offline. For instance, the COACH system helped people suffering from Dementia by giving them appropriate prompts as needed in their daily activities [2]. In this system, there is a single fixed goal of washing hands for the user. In *Electric Elves*, the assistant is used to reschedule a meeting should it appear that the user is likely to miss it [6]. These systems do not have a hierarchical goal structure for the user while in our system, the assistant infers the user's goal combinations and renders assistance.

Several plan recognition algorithms use a hierarchical structure for the user's plan. These systems would typically use a hierarchical HMM [17] or an abstract HMM [1] to track the user's plan. They unroll the HMMs to a DBN and perform inference to infer the user's plan. We follow a similar approach, but the key difference is that in our system, the user's goals are relational. Also, we allow for richer models and do not restrict the user's goal structure to be modeled by a HMM. We use the qualitative influence statements to model the prior over the user's goal stack. We observe that this could be considered as a method to incorporate richer user models inside the plan recognition systems. There has been substantial research in the area of user modeling. Systems that have been used for assistance in spreadsheets [7] and text editing [8] have used handcoded DBNs to infer about the user. Our system provides a natural way to incorporate user models into a decision-theoretic assistant framework.

In recent years, there have been several first-order probabilistic languages developed such as PRMs [14], BLPs [15], RBNs [12], MLNs [13] and many others. One of the main features of these languages is that they allow the domain expert to specify the prior knowledge in a succinct manner. These systems exploit the concept of parameter tying through the use of objects and relations. In this paper, we showed that these systems can be exploited in decision-theoretic setting. We combined the hierarchical models typically used in reinforcement learning with the kinds of influence knowledge typically encoded in relational models to provide a strong bias on the user policies and accelerate learning.

6 Conclusions and Future Work

In this work we proposed the incorporation of parameterized task hierarchies to capture the goal structure of a user in a decision-theoretic model of assistance. We used the relational models to specify the prior knowledge as relational hierarchies and as a means to provide informative priors. We evaluated our model against the non-hierarchical and non-relational versions of the model and established that combining both the hierarchies and relational models makes the assistant more useful. The incorporation of hierarchies would enable the assistant to address several other problems in future. The most important one is the concept of parallel actions. Our current model assumes that the user and the assistant have interleaved actions and cannot act in parallel. Allowing parallel actions can be leveraged if the goal structure is hierarchical as the user can achieve a subgoal while the assistant can try to achieve another one. Yet another problem that could be handled due to the incorporation of hierarchies is the possibility of the user changing his goals midway during an episode. Finally, we can also imagine providing assistance to the user in the cases where he forgets to achieve a particular subgoal.

Acknowledgements

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010.

References

1. Bui, H., Venkatesh, S., West, G.: Policy recognition in the Abstract Hidden Markov Models. In: JAIR, vol. 17 (2002)
2. Boger, J., Poupart, P., Hoey, J., Boutilier, C., Fernie, G., Mihailidis, A.: A Decision-Theoretic Approach to Task Assistance for Persons with Dementia. In: IJCAI 2005, pp. 1293–1299 (2005)
3. Ambite, J.L., Barish, G., Knoblock, C.A., Muslea, M., Oh, J., Minton, S.: Getting from Here to There: Interactive Planning and Agent Execution for Optimizing Travel. In: IAAI, pp. 862–869 (2002)
4. Myers, K., Berry, P., Blythe, J., Conleyn, K., Gervasio, M., McGuinness, D., Morley, D., Pfeffer, A., Pollack, M., Tambe, M.: An Intelligent Personal Assistant for Task and Time Management. AI Magazine (June 2007)
5. Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-Dynamic Programming. Athena Scientific (1996)
6. Varakantham, P., Maheswaran, R., Tambe, M.: Exploiting belief bounds: Practical POMDPs for personal assistant agents. In: Kudenko, D., Kazakov, D., Alonso, E. (eds.) AAMAS 2004. LNCS (LNAI), vol. 3394, Springer, Heidelberg (2005)
7. Horvitz, E., Breese, J., Heckerman, D., Hovel, D., Rommelse, K.: The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In: UAI, pp. 256–265 (1998)

8. Hui, B., Boutilier, C.: Who's asking for help?: a Bayesian approach to intelligent assistance. In: Boutilier, C. (ed.) IUI, pp. 186–193 (2006)
9. Fern, A., Natarajan, S., Judah, K., Tadepalli, P.: A Decision-Theoretic Model of Assistance. In: IJCAI (2007)
10. Dietterich, T.G.: Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. In: JAIR, vol. 13, pp. 227–303 (2000)
11. Tadepalli, P., Givan, R., Drissens, K.: Relational Reinforcement Learning - An Overview, Workshop on Relational Reinforcement Learning. In: ICML (2004)
12. Jaeger, M.: Relational Bayesian Networks. In: UAI 1997
13. Domingos, P., Richardson, M.: Markov logic networks. Mach. Learn. 62(1-2), 107–136 (2006)
14. Getoor, L., Friedman, N., Koller, D., Pfeffer, A.: Learning Probabilistic Relational Models. In: Dzeroski, S., Lavrac, N. (eds.) Invited contribution to the book Relational Data Mining (2001)
15. Kersting, K., De Raedt, L.: Bayesian Logic Programs. In: Cussens, J., Frisch, A.M. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, Springer, Heidelberg (2000)
16. Murphy, K., Paskin, M.: Linear time inference in hierarchical HMMs. In: NIPS (2001)
17. Fine, S., Singer, Y., Tishby, N.: The Hierarchical Hidden Markov Model: Analysis and Applications. Machine Learning 32(1), 41–62 (1998)
18. Getoor, L., Grant, J.: A Probabilistic Relational Language, Machine Learning Journal, 2005. Machine Learning 62(1-2), 7–33 (2006)
19. Natarajan, S., Tadepalli, P., Altendorf, E., Dietterich, T.G., Fern, A., Restificar, A.: Learning First-Order Probabilistic Models with Combining Rules. In: Proceedings of ICML 2005 (2005)

Using Bayesian Networks to Direct Stochastic Search in Inductive Logic Programming

Louis Oliphant and Jude Shavlik

Computer Sciences Department, University of Wisconsin-Madison

Abstract. Stochastically searching the space of candidate clauses is an appealing way to scale up ILP to large datasets. We address an approach that uses a Bayesian network model to adaptively guide search in this space. We examine guiding search towards areas that previously performed well and towards areas that ILP has not yet thoroughly explored. We show improvement in area under the curve for recall-precision curves using these modifications.

1 Introduction

Inductive Logic Programming (ILP) [3] algorithms search for explanations written in first-order logic that discriminate between positive and negative examples. Two open challenges for scaling ILP to larger domains include slow evaluation times for candidate clauses and large search spaces in which to find those clauses. Our work addresses this second challenge using adaptive stochastic search.

Algorithms such as Progol [6] and Aleph [12] also address the second challenge by constraining the size of the search space using a bottom clause constructed from a positive seed example. A bottom clause is constructed from a positive seed by using a set of user-provided modes. The user creates a mode for each literal in the background knowledge. Modes indicate which arguments of the literal are input arguments and which are output arguments. As the bottom clause is being constructed, only literals whose input arguments are satisfied by the output arguments of literals already in the bottom clause may be added. The modes create a dependency between the literals of a bottom clause. A literal may not be added to a candidate clause unless its input arguments appear as output arguments in some prior literal already in the candidate clause.

Even with these constraints the size of the search space usually is much larger than can be exhaustively searched. Zelezny *et al.* [13] have incorporated a randomized search algorithm into Aleph in order to reduce the average search time. Their rapid random restart (RRR) algorithm selects an initial clause using a pseudo-uniform distribution and performs local search for a fixed amount of time. This process is repeated for a fixed number of tries.

Our work builds on top of the RRR algorithm and bottom-clause generation. We construct a non-uniform probability distribution over the search space that biases search towards more promising areas of the space and away from areas which have already been explored or that look unpromising. We use a pair of

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 191–199, 2008.

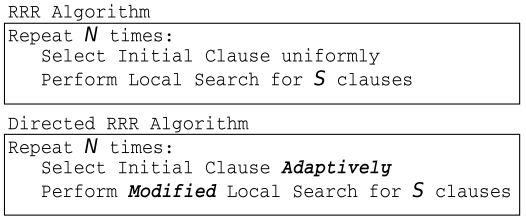


Fig. 1. Pseudo-code showing the Rapid Random Restart (RRR) algorithm and our modified version of RRR

Bayesian networks to capture this skewed distribution. The structure of the networks is determined by the bottom clause and the parameters are trained as ILP's search progresses. The clauses that are evaluated by the ILP system become the positive and negative examples for training the Bayesian networks. The trained networks are then used to select the next initial clause and to modify the local search portion of RRR.

2 Directed Stochastic Search Algorithm

Zelezny's RRR algorithm appears in Figure 1 on the top. We have incorporated a non-uniform distribution into this algorithm in order to bias search towards more promising areas of the search space. Our modifications appear in Figure 1 on the bottom. In the following subsections we explain our method of modeling a probability distribution over ILP's search space, training the parameters of the model, and using this trained model to modify RRR's search process.

2.1 Modeling ILP's Search Space with Bayesian Networks

A Bayesian network [5] is a directed acyclic graphical model that captures a full joint probability distribution over a set of variables. Each node in the graphical model represents a random variable. Arcs represent dependencies between variables. Each node has a probability distribution showing the probability of the variable given its parents.

ILP's search space consists of subsets of literals from the bottom clause. A sample bottom clause appears in Figure 2 on the left. Literals' arguments have been annotated with +/- marks to indicate input/output arguments taken from the user-provided modes. Not all subsets of literals are legal clauses. A clause is legal and in the search space if the input arguments of each literal in a clause first appear as output arguments for some literal earlier in the clause.

We capture these dependencies created by the user-provided modes in a graphical model. Figure 2 on the right shows graphically the dependencies found in the bottom clause. Each node in the network represents a Boolean variable that is true if the corresponding literal from the bottom clause is included in a candidate clause. Each arc represents a connection between the output arguments

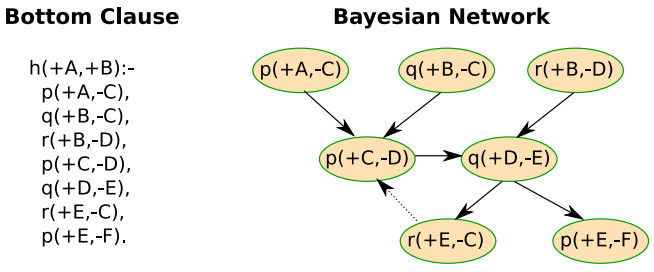


Fig. 2. A portion of a Bayesian network built from the literals in a bottom clause. The + marks indicate input arguments and the - marks indicate output arguments taken from the user-provided modes. The head literal is not part of the Bayes net. Arcs indicate dependencies between the output variables of one literal to the input variables of another literal. Dotted arcs are dropped to maintain the acyclic nature needed for Bayesian networks.

of one literal to the input arguments of another literal. Dotted arcs indicate dependencies that are dropped in order to maintain the acyclic nature needed for Bayesian networks. The structure of the Bayesian network is determined by the bottom clause while the parameters are learned as ILP's search progresses.

The algorithm to create the Bayesian network structure from a bottom clause appears in Figure 3. The algorithm constructs the Bayes net in a top-down approach. Variable `group` contains all literals whose input variables are satisfied by the `Head` literal or any literal already in the network. The literals in `group` are added one at a time to the Bayes net. As the literal is added into the Bayes net the algorithm connects it to all literals that contain some input variable that is not contained by the `Head` literal. Creating a group of literals and adding them to the network repeats until all literals have been added.

2.2 Training the Model

After creating the Bayesian network structure, we still need to learn the parameters of the model. Each node contains a conditional probability table (CPT) that predicts the probability the node is true given its parents. We estimate these probabilities from training data collected during ILP's search.

We construct two networks that have the same graphical structure. The parameters of the first network are trained using "good" clauses seen during search, while the parameters of the second are trained on all clauses evaluated. These two networks provide probabilities that indicate, respectively, how good a candidate clause is and how similar the clause is to past clauses. These distributions are density estimators indicating the promising areas of the search space and which areas have already been explored.

The parameters of a node are estimated using ratios of weighted counts between clauses that contain the literal and those that do not for the various

```
function CONSTRUCT_NETWORK(⊥): returns a Bayesian network
input: ⊥, a bottom clause consisting of a Head and Body
bayes_net=empty
reached=input variables from Head
while Body is not empty do:
    group={l|l ∈ Body and l's input variables are in reached}
    for each lit ∈ group do:
        ADD_NODE(lit,bayes_net) /* connects to lit all nodes
                                   in bayes_net that satisfy an input
                                   variable of lit*/
    Body = Body - group
    reached = reached + output variables from group
return bayes_net
```

Fig. 3. Pseudo-code showing the construction of a Bayesian network from a bottom clause

settings of the parents. We update the parameters during search when a clause is evaluated on the training data.

The first network, which estimates the probability that a clause is "good," is trained using high-scoring clauses. We have tried several methods for deciding which clauses to use. Our current approach involves using the Gleaner algorithm [4]. Gleaner retains a set of high-scoring clauses across a range of recall values. All clauses that are retained in Gleaner's database are used, along with those clauses in the trajectory from the initial clause to the one retained in the database. We use weighted counts (a clause's F1 score is its weight) with higher-scoring clauses receiving higher weights. This allows better clauses to have more influence on the network.

The second network, which estimates the probability that a new clause is similar to past clauses, is trained on all clauses considered, using a uniform weight on the clauses. The combination of the probabilities from these two networks will allow us to trade off exploration of unvisited portions of the hypothesis space for exploitation of the promising portions.

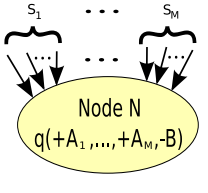


Fig. 4. Node with many arguments

We have found that some nodes in our networks have 20 or more parents. In order to reduce the size of the conditional probability tables, we utilize a noisy-OR assumption [7]. The noisy-OR model assumes that all parents of a node are independent of each other in their ability to influence the node. This reduces the size of the CPT to be linear in the number of parents.

Figure 4 shows a single node whose corresponding literal has several input arguments. Each argument may be satisfied by one of many parents. We calculate the probability that a node, N, is true using the formula

$$P(N = t|\Pi(N)) = \prod_{j=1}^{M} P(N = t|S_j(N)) = \prod_{j=1}^{M} \left(1 - \prod_{R \in S_j(N)} P(N = f|R)\right)$$

where $\Pi(N)$ are the parents of N and S_j is the subset of $\Pi(N)$ that satisfy input argument j. The outer product ranges over all M input variables of the node. This reduces the conditional probability to a product of simpler conditional probabilities, one for each input argument. The simpler conditional probabilities are modeled as noisy-ORs. $P(N = f|R = f)$ is set to equal 1 so if any input argument is not satisfied by at least one parent then that portion of the product, $P(N|S_j(N))$, will be zero, making the entire product zero. This limits the clauses that have a non-zero probability to those that are legal.

2.3 Using the Model to Guide Search

The probabilities provided by the Bayesian networks are incorporated into a weight that we can attach to clauses. Recall that two networks are created. We call the probability from the network trained on "good" clauses $EXPLOIT$ and the probability from a second network trained on all clauses $EXPLORED$. We combine these two estimates into a weight for a candidate clause using the formula

$$W = \alpha * EXPLOIT + (1 - \alpha) * (1 - EXPLORED)$$

where $0 \leq \alpha \leq 1$. We can then set parameter α to trade-off exploration for exploitation. In order to interleave exploration and exploitation we select α from a range of values each time an initial clause is selected.

We use the clause weight to modify the RRR algorithm in two ways. The original RRR algorithm selects an initial clause uniformly. Our modified version of RRR performs K hill-climbing runs using the weights generated by the Bayesian network to guide search. We then select a single initial clause from the K local peaks by selecting a clause found at one of these peaks. We sample proportional to the weights of the clauses, with the idea that the search will begin in a higher-scoring and more diverse area of the space. Assigning a weight to a clause is relatively fast compared to evaluating the clause on the training data.

Next the original RRR algorithm performs a local search around this initial clause, expanding the highest-scoring clause on the open list and evaluating *all* of its neighbors on the training set. Our modified version of RRR attaches a weight to the neighboring clauses before they are evaluated on the training set and only a *high-weighted subset* of size L are retained and evaluated. This reduces the number of clauses that are evaluated on the training data which are close to any one initial clause, thus broadening the search and guiding it to areas of the search space which are more likely to contain high-scoring, unique clauses.

Our algorithm interleaves optimizing using our model and optimizing using real data. Assigning a weight to a clause using our model is much faster than evaluating a clause on real data when the dataset is large. We hypothesize our approach will outperform standard RRR search in terms of area under the

recall-precision curve, when we allow RRR and our modified version to each evaluate the same number of clauses on real training data.

3 Directed-Search Experiments

We compare our search modifications using the Gleaner algorithm [4] of Goadrich *et al.* Following their methodology we compare area under the recall-precision curves (AURPC) using Gleaner with the standard RRR search algorithm and with our modified RRR search algorithm.

We evaluated our modifications to the RRR search algorithm on three datasets. The protein-localization biomedical information extraction dataset [4] contains 7,245 sentences from 871 abstracts taken from the Medline database. There are over 1,200 positive phrase-phrase relations and a positive:negative ratio of over 1:750. The relations were marked by hand using inter-expert agreement to reduce the number of incorrectly marked relations.

The gene-disorder dataset also comes from the biomedical information extraction domain. The dataset originally comes from work by Ray and Craven [9]. Due to memory limitations we scale the dataset by downsampling the abstracts. The relations in the dataset are marked by a computer algorithm. Our scaled down version contains 239 positive and 103,959 negative examples.

Finally, the University of Washington dataset contains 113 positive examples and 2,711 negative examples [10]. The task is to learn the advisor-student relationship. Background information includes facts about papers published and classes taught.

We assigned the α parameter to be between 0.01 and 0.75 in order to encourage exploration. The K parameter controlling the number of hill-climbing runs for each initial clause was set to ten, and the L parameter controlling how many neighbors of a clause are retained was set to twenty. The internal parameters of the Bayesian networks were updated as clauses were evaluated. We ran our experiment using 100 seeds for the two information extraction task and 50 seeds for the advisor-student task. We evaluated performance after one thousand, ten thousand, and twenty-five thousand clauses per seed.

Figure 5 shows the AURPC versus the number of clauses evaluated averaged over all five folds of each dataset. Although the improvement is small in the protein-localization task, it is significant for the first two points at the 95% confidence level using a paired t test. We also show improvement in the advisor-student task, however this improvement is not signification. Perhaps this is because the dataset is smaller, which may cause larger variance between folds. On the gene-disorder task no improvement is found. One possible reason for this may stem from the fact that the dataset has a considerable amount of noise due to the automatic labeling of the data as documented by Ray and Craven [9]. One additional area that may improve performance would be to use a tuning set for setting the algorithms parameters.

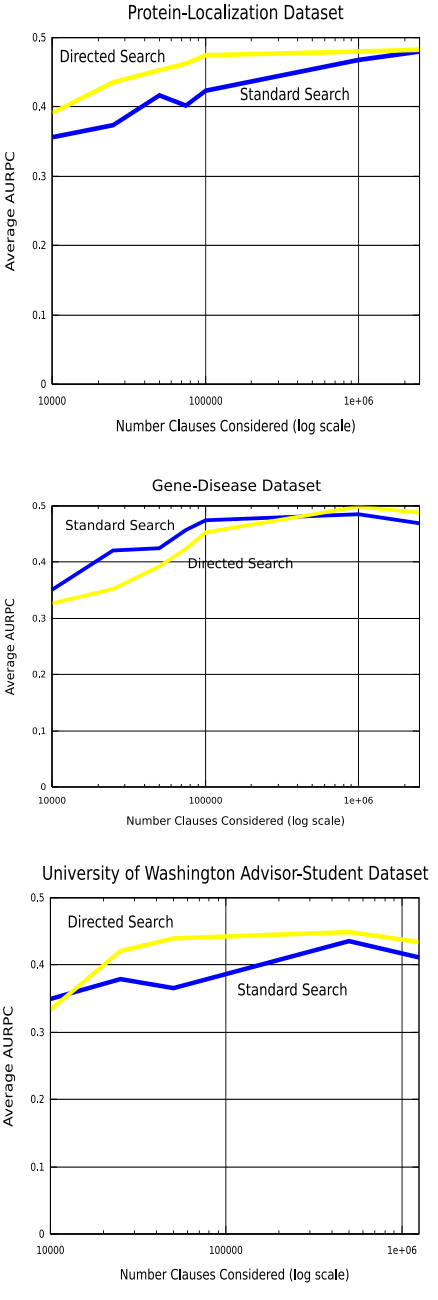


Fig. 5. Comparison on three datasets of AURPC for varying number of clauses considered using Gleaner with and without a directed RRR search algorithm

4 Related Work

Several other researchers have used models to guide the search process. Pelikan *et al.* [8] developed the Bayesian Optimization Algorithm (BOA) which learns the structure and parameters of a Bayesian network from sampling the search space and uses the learned model to select a higher-scoring sample. Rubinstein's cross-entropy (CE) algorithm [11] comes from the rare-event modeling domain. It begins by uniformly sampling the search space and then the sample is sorted and a model is built using the highest ρ-percentile of the sample.

The STAGE algorithm by Boyan and Moore [1] learns an evaluation function that predicts the outcome of some type of local search, such as hill-climbing or simulated annealing. Their algorithm iterates between optimizing on real data and optimizing on the learned model. One final example is taken from the ILP literature. DiMaio and Shavlik [2] built a neural network to predict a clause's score in ILP. They use the clause's predicated score to modify selection of the initial clause, as well as to order clauses on the open list.

5 Conclusions and Future Work

Stochastic search of the space of clauses provides a means for ILP to scale to larger datasets. Our basic approach is to convert the dependency structure found in Aleph's modes into two Bayesian networks, whose parameters are trained as search progresses. These networks are used to influence where in the space of clauses will be searched next. We have shown improvement on area under the recall-precision curve experiments on two of three datasets by using this adaptive stochastic approach.

We plan on improving this approach by refining the structure of the networks as search progresses. An alternative approach, designing undirected graphical models that do not need to drop dependencies between literals, will also be considered. Fitting the parameters of either type of model quickly is important to reduce the amount of search. Allowing transfer of parameters from a model trained using one seed to models trained on some other seed will reduce the amount of training time needed for the new model. We plan on designing a mechanism for transferring these parameters. Finally we plan on including mechanisms to search for diverse clauses more directly, only allowing a clause to be retained when it is different enough from previously retained clauses. This should improve the diversity of the set of clauses, viewing the set more as an ensemble than as a single theory.

Acknowledgements

We would like to thank Mark Goadrich, Jesse Davis, Trevor Walker, and the anonymous reviewers for comments and suggestions on this work. This project is funded by DARPA IPTO under contract FA8650-06-C-7606 and DARPA grant HR0011-04-1-0007.

References

[1] Boyan, J., Moore, A.: Learning Evaluation Functions for Global Optimization and Boolean Satisfiability. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence, pp. 3–10 (1998)

[2] DiMaio, F., Shavlik, J.: Learning an Approximation to Inductive Logic Programming Clause Evaluation. In: Proceedings of the 14th International Conference on Inductive Logic Programming, Porto, Portugal, pp. 80–97 (2004)

[3] Džeroski, S., Lavrac, N.: An Introduction to Inductive Logic Programming. In: Džeroski, S., Lavrac, N. (eds.) Proceedings of Relational Data Mining, pp. 48–66. Springer, Heidelberg (2001)

[4] Goadrich, M., Oliphant, L., Shavlik, J.: Gleaner: Creating Ensembles of First-Order Clauses to Improve Recall-Precision Curves. Machine Learning 64(1-3), 231–261 (2006)

[5] Heckerman, D.: A Tutorial on Learning with Bayesian Networks. Technical Report MSR-TR-95-06, Microsoft Research, Redmond, Washington, (revised June 1996)

[6] Muggleton, S.: Inverse Entailment and Progol. New Generation Computing Journal 13, 245–286 (1995)

[7] Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc. San Francisco (1988)

[8] Pelikan, M., Goldberg, D., Cantú-Paz, E.: BOA: The Bayesian Optimization Algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, FL, vol. I, pp. 525–532. Morgan Kaufmann Publishers, San Francisco (1999)

[9] Ray, S., Craven, M.: Representing Sentence Structure in Hidden Markov Models for Information Extraction. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence (2001)

[10] Richardson, M., Domingos, P.: Markov Logic Networks. Machine Learning 62(1-2), 107–136 (2006)

[11] Rubinstein, R., Kroese, D.: The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization. In: Monte-Carlo Simulation and Machine Learning, Springer, Secaucus (2004)

[12] Srinivasan, A.: The Aleph Manual Version 4. (2003), http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/

[13] Železný, F., Srinivasan, A., Page, D.: Lattice-Search Runtime Distributions be Heavy-Tailed. In: Matwin, S., Sammut, C. (eds.) ILP 2002. LNCS (LNAI), vol. 2583, pp. 333–345. Springer, Heidelberg (2003)

Revising First-Order Logic Theories from Examples Through Stochastic Local Search

Aline Paes[1], Gerson Zaverucha[1], and Vitor Santos Costa[2]

[1] Department of Systems Engineering and Computer Science - COPPE
Federal University of Rio de Janeiro (UFRJ), Brazil
{ampaes, gerson}@cos.ufrj.br
[2] LIACC and DCC/FCUP, Universidade do Porto, Portugal
vsc@dcc.fc.up.pt

Abstract. First-Order Theory Revision from Examples is the process of improving user-defined or automatically generated First-Order Logic (FOL) theories, given a set of examples. So far, the usefulness of Theory Revision systems has been limited by the cost of searching the huge search spaces they generate. This is a general difficulty when learning FOL theories but recent work showed that Stochastic Local Search (SLS) techniques may be effective, at least when learning FOL theories from scratch. Motivated by these results, we propose novel SLS based search strategies for First-Order Theory Revision from Examples. Experimental results show that introducing stochastic search significantly speeds up the runtime performance and improve accuracy.

1 Introduction

A variety of Inductive Logic Programming systems have been developed to automatically learn First-Order Logic (FOL) theories [11], [4], with good results on a number of important applications [13,12], [3]. Most such systems are designed to learn theories from scratch, given a set of examples and a fixed body of prior knowledge, the *background knowledge*. There has been relatively less work on the problem of *repairing* incorrect or incomplete theories. One example of theories that could be repaired or improved are theories that had been elicited from a domain expert, and thus may include useful information, but on the other hand may be incomplete, and/or rely on incorrect assumptions, or even be inconsistent. A second common example is the case where *new* examples are not well explained by the original theory. In such cases standard ILP systems would take one of the two following positions: they could either discard the initial theory, or consider it as part of background knowledge which can not be modified.

Since the task of knowledge acquisition is difficult and time-consuming, and since the original theory may contain valuable prior information, one would like to take advantage of the original theory as a start point to the learning process.

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 200–210, 2008.

Ideally, this should accelerate learning time and result in more accurate theories. *Theory refinement* systems have been proposed towards this goal [15,7]. Such systems assume the initial theory is approximately correct. If so, then only some *points* in the theory prevent it from correctly modeling the dataset. The idea is therefore to search for such points in the theory and *revise* them instead of using an algorithm that learns a whole new theory from scratch. Note that these revision algorithms can be seen as a generalization of learning from scratch, as performed by most ILP systems. However, in contrast to most ILP systems, theory revision algorithms do not apply cover removal, where clauses that explain uncovered examples are searched sequentially. Instead, theory revision algorithms perform search in the space of whole theories. Arguably, cover removal frequently generate unnecessarily long hypothesis with too many clauses.

Theory revision systems operate by searching for revision points, that is, the points which explain faults in the theory, and then proposing revisions to such points, through applying at each point a number of matching *revision* operators. Therefore, theory revision can be seen as a search process, and very much as most ILP algorithms, revising logic programs may need to search a very large search space and therefore may incur big time and storage requirements. Search space grows quickly with the size of the knowledge base. We would also expect for search to be harder if the theory has more faults. Last, theory revision systems are particularly ambitious in that they tackle whole theories, which is known to be a hard problem [1].

One possible way of alleviating the huge requirements of searching in theory revision algorithms is to take advantage of clever search strategies such as stochastic local search (SLS). Such methods have been successfully applied to solve difficult combinatorial propositional problems [8,10,9] and recently they have also been applied to learn theories from scratch in ILP systems [5,14], substantially improving the efficiency of both domains. Motivated by these works and by the increased combinatorial explosion of searching in entire theories, we investigate the relevance of applying SLS in a theory revision algorithm. To do so, we develop algorithms that performs stochastic local search, when proposing revisions and when searching for a revision to be implemented. Such stochastic theory revision approach is compared to a theory revision algorithm that performs only greedy hill-climbing search and to a state-of-art ILP system.

The outline of the paper is as follows. Some preliminary knowledge concerning SLS and theory revision are reviewed in sections 2 and 3, respectively. The algorithms developed to revise FOL theories from examples through SLS are devised in section 4. Experimental results are presented in section 5, followed by conclusions and future work in section 6.

2 Stochastic Search

Stochastic Search algorithms are a family of search algorithms that strongly rely on randomized decisions while searching for solutions. Stochastic Local search

algorithms (SLS) are based on local search techniques. They therefore abandon completeness in favor of trying to achieve the best exploitation of bounded resources [8]. One major motivation and successful application of SLS has been in satisfiability checking of propositional formulae, namely through the well-known GSAT [10] and WalkSAT [9] algorithms. A large number of tasks in areas such as planning, scheduling and constraint solving can be encoded as a satisfiability problem, and empirical observations show that SLS often can substantially improve their efficiency [2,8].

Several machine learning algorithms can be described as search algorithms. There has therefore been some interest in applying SLS and related techniques to this area. Chisholm and Tadepalli [2] used stochastic search to perform rule learning in their system LERILS and compare its performance to other learning algorithms, with encouraging results. Again in the area of rule learning, Rückert et.al. present an SLS algorithm for learning k-term DNFs, that is, theories with at most k clauses [8]. They proposed a novel SLS algorithm specific for this task, and evaluated its performance with excellent results.

Stochastic Search in ILP Several Inductive Logic Programming algorithms perform search on a vast search spaces, and most of them do include a limited amount of stochastic search. As an example, Progol-like systems randomly select examples as *seeds* to start their search [4,11]. It is therefore unsurprising that research on stochastic search has taken place since early ILP days [14].

A recent study takes this point further by implementing and evaluating the performance of several randomization strategies in the ILP system Aleph [14]. The authors use a deterministic general-to-specific search as reference. They then compare a variety of randomized algorithms. The stochastic strategies were framed in terms of a single clause search algorithm. The results indicate that the randomized search strategies outperforms in terms of search space deterministic clause search across large intervals of the parameter space, and motivates further research across ILP.

3 First-Order Logic Theory Revision

First-order theory revision is a challenging subject, particularly complex because we do not revise single clauses; instead we must deal with the issues arising from including a theory with multiple clauses. Although our studies could be performed on any theory revision algorithm, we choose the FORTE (First Order Revision of Theories from examples) [7] system to implement and experimentally evaluate our approach. FORTE performs hill-climbing search through a space of both specialization and generalization operators in an attempt to find a minimal revision to a theory that makes it consistent with the set of training examples. The top-level algorithm is exhibited as Algorithm 1. The key ideas are:

1. Identify all the revision points in the current theory.
2. Generate a set of proposed revisions for each revision point starting from the one with the highest potential and working down the list. *Potential* is defined as the number of misclassified examples that could be turned into correctly classified from a revision in that point. The revisions are proposed through a number of revision operators. In this work we consider *Delete-rule* and *Add-antecedent* as specializations operators and *Delete-antecedent* and *Add-rule* as generalization operators. [1]
3. Score each revision through the actual increase in theory accuracy it achieves.
4. Retain the revision which most increases the score.

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

Algorithm 1. FORTE Algorithm (Richards and Mooney, 1995)

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

4 Stochastic First-Order Logic Theory Revision

Previous research on stochastic search for ILP has focused on clause search starting from the empty theory. Next, we study whether stochastic search can improve performance on the theory-level search performed by theory revision systems. Following WalkSAT and related SLS algorithms, the approaches we propose perform a local, randomized-walk search, alternating between stochastic and greedy moves with the type of the move to be executed being chosen according to a prior probability p.

[1] There are others operators not used here, such as *predicate invention* and *abduction*. Actually, any operator used in first-order machine learning can be used in a theory revision system.

As observed above, a first major difference between our problem and prior research is that we need to randomize search over theories. A second major difference between our problem and prior approaches is that we would not like to start from a random or empty hypothesis. Instead, we would like to take advantage of an initial theory provided to the system. In order to choose the revision to be implemented, theory revision systems such as FORTE proceed in two steps. First, the system searches for proposed revisions through considering all revision operators at all revision points. This process can be quite expensive as some operators must add antecedents, which requires searching through all possible goals in the database. Next, the revision system chooses the revision with highest score. This argues for two different phases where one could take advantage of a randomized strategy:

1. Operator Search: as operator search is dominated by antecedent search, we may benefit from randomizing antecedent search.
2. Revision search: instead of considering all possible revisions, we may randomize the possible revisions to be implemented.

4.1 Stochastic Local Search for Antecedents

Except for the *Delete-Rule* operator, all revision operators must *search* the best antecedent to add/delete. This suggests using stochastic methods in order to explore the search space in a more efficient way. Next, we investigate a hill-climbing stochastic search. We believe that a greedy algorithm could lead to either too large or too small clauses, as it would run for a fixed number of steps. Therefore, the search strategies devised here always execute until adding/deleting more antecedents cannot improve the score. The stochastic version of algorithms for adding or deleting antecedents is exhibited in Algorithm 2. Such algorithm is executed into the operations in line 5 in Algorithm 1. FORTE provides two separate algorithms for producing a specialized clause: hill-climbing antecedent addition and relational pathfinding. In this work we focus on hill-climbing, as it is more suitable to most datasets we consider. It is important to notice that the delete-antecedent operator benefits less from stochastic local search than add-antecedent, since the search space is restricted to goals in the clause, and is therefore much smaller.

The process of adding or deleting antecedents using a SLS component starts by getting all antecedents that could be added (deleted) in (from) a clause chosen as revision point. If the antecedents are being added, the algorithm must generate all possible antecedents from the database. If they are being deleted, it is enough to collect the antecedents from the clause. As usual, we can execute a stochastic or a greedy move, depending upon a fixed probability $p1$ (addition) or $p2$ (deletion). In a stochastic move, an antecedent is chosen at random and then scored. If this antecedent improves the current score it is added (deleted) in (from) the clause. Otherwise, another antecedent will be chosen at random. If the move is greedy, all the antecedents are scored and the one with the highest score is chosen.

Algorithm 2. Algorithm for adding/deleting antecedents using hill-climbing SLS

```
repeat
   get all antecedents;
   with probability p1/p2:
      choose an antecedent at random whose score improves the current one, and,
       add/delete it to/from the clause;
   otherwise:
      for each antecedent
         calculate score;
      add/delete to/from the clause the antecedent with the highest score if it
       improves the current score of the clause;
until no antecedent can improve the score;
```

4.2 Stochastic Local Search for Revisions

We propose two algorithms for randomizing revision search. The first algorithm, *greedy*, implements greedy stochastic local search. The algorithm stops either when it reaches a maximum number of steps or when it reaches a maximum score. The second algorithm, *hill-climbing* chooses a move at random if it *improves* the score, and only if so. The *hill-climbing* algorithms stops when further revisions cannot improve the score in the same way FORTE does. The greedy and hill-

Algorithm 3. A greedy SLS theory revision Algorithm

```
1. while score < maxScore and steps < maxSteps
2.    generate revision points;
3.    with probability p
4.       list all possible revisions from the generated revision points;
5.       next_revision = a revision chosen at random from the list of possible
                         revisions;
6.    otherwise
7.       FORTE Generate and Search revisions procedure
8.       next_revision = best revision;
9.    implements next_revision;
10.   steps ++;
```

climbing SLS revision theory algorithms are presented in detail as Algorithms 3 and 4, respectively, where *FORTE Generate and Search revisions procedure* corresponds to lines 3 to 7 in Algorithm 1. As usually in SLS methods, the algorithms do a random move with probability p or do a FORTE-like move with probability $1 - p$.

The two algorithms differ on the random move and on how to terminate. In the greedy algorithm, a revision is just chosen at random. In the hill-climbing algorithm, a revision is chosen at random but is only accepted if it actually improves the score. The two algorithms also differ on their stopping criteria. The

greedy algorithm stops on finding a best solution and on time. The hill-climbing algorithm stops if finding that no revision can actually improve the score, just like FORTE does. Thus, Algorithm 4 replaces the lines 3-9 in Algorithm 1 while Algorithm 3 replaces the whole algorithm only because the stopping criteria in this last case is different from the original FORTE.

Notice that we do not need to explicitly enumerate all possible revisions. In fact, given a revision point, we know that it either contributes to the misclassification of positives, and is therefore a generalization point, to the misclassification of negatives, and is therefore a specialization point, or to the misclassification of both negatives and positives, and is therefore both. Thus, given all revision points and their types, we can estimate the number of possible revisions and select one at random.

Algorithm 4. A hill-climbing SLS theory revision Algorithm

```
1. with probability p:
2.     list all possible revisions from the generated revision points;
3.     next_revision = a revision chosen at random from the list of possible revisions
                       whose score > current ;
4. otherwise:
5.     FORTE Generate and Search revisions procedure
6.     next_revision = best revision;
7. if next_revision improves the theory
8.     implement next_revision;
```

5 Experimental Results

We performed some experiments in order to evaluate the different approaches using three ILP benchmarks. The set of experiments aims to answer two main questions:

1. Does the stochastic revision approach decrease the running time of the standard revision approach while maintaining or improving the predictive accuracy?
2. Is it possible to improve theories obtained from the state-of-art ILP system Aleph, escaping from local minima in a reasonable time by applying stochastic revision algorithms?

Algorithms In this work we analyse the following algorithms. **(a)** The original FORTE; **(b)** Greedy stochastic revision search plus stochastic antecedents search (Algorithm 3 plus Algorithm 2); **(c)** Hill-climbing stochastic revision plus stochastic antecedent search (Algorithm 3 plus Algorithm 2).

We use Aleph [11], a state-of-art ILP system, as the reference system. Initial theories were obtained from two experiments:

1. In each fold of a 10 fold cross-validation procedure using 50% of the examples a theory is generated by Aleph (45% of the examples in the training data

5% in the test). Then each one of these theories returned by each fold are revised by the SLS algorithm and respective folds from all examples of the dataset. That is SLS revises the theory obtained from Aleph using the whole dataset (obviously without the 10% of the testset).
2. In each fold of a 10 fold cross-validation procedure a theory is generated by Aleph from all the examples in the dataset. Then each one of these theories are revised considering their respective fold (i.e., the same folds are used to generate and revise the theories).

Experimental methodology. We use the following datasets. *Mutagenesis* is a well-known domain for predicting structure-activity relationship (SAR). We use here the "regression friendly" dataset, as discussed in [13], composed of 188 positive examples and 42 negative examples. *Carcinogenesis* is another well-known domain from SAR for predicting carcinogenic activity in rodent bioassays [12], with 162 positive examples and 136 negative examples. *Alzheimer*, compares analogues of Tacrine, which is a drug against Alzheimer's disease [3]. In this work we used only the dataset related to the inhibit **amine** reuptake property, composed of 343 positives examples and 343 negative examples.

The experiments were conducted on Pentium4 machines with 1 Gb of RAM (Mutagenesis), Pentium Dual Core with 2Gb of RAM (Carcinogenesis) and on a Cluster with Pentium HT machines (Alzheimer). [2] All the algorithms were run using 10-fold cross validation; additionally, the stochastic algorithms were run 25 times. All the probability parameters were defined as 50%. The maximum number of steps for algorithm 3 were defined as 5 for Mutagenesis and 10 for Carcinogenesis and Alzheimer_amine, since these two last ones are more complex domains. [3]The hypothesis were evaluated through their accuracy, which is the same evaluation function used in original FORTE.

Results. Tables 1 and 2 show the values of accuracy and run time returned by theory revision stochastic algorithms, Aleph and original FORTE. In such tables, "SLS HC" means that the stochastic search for revision is hill-climbing and "SLS greedy" means that the stochastic search for revisions is greedy. As stated before both algorithms randomize the search for antecedents. Notice that we always show the the runtime of the revision algorithms considering the whole dataset and the predictive accuracy of all algorithms considering the same test set. The symbol $\star$ means that the stochastic algorithm makes a statistically significant improvement in Aleph and $\bullet$ indicates an improvement over original FORTE, both considering 95% of significance level. For Alzheimer_amine we run the revision algorithms with and without the relational pathfinding search for antecedents (the last row of the tables show the results without relational pathfinding).

[2] We had to use three different machines (one machine for each dataset) due to resource limitations. Note that the algorithms were always evaluated in the same machine for each dataset.

[3] It was not our intention to use the best probability and number of steps parameters. Future work could include using validation sets in an internal cross-validation procedure to determine the best ones.

Table 1. Runtime of Aleph, FORTE and stochastic revision algorithms

Dataset	Method	Aleph runtime	Forte runtime	SLS HC runtime	SLS greedy runtime
Mutag.	1	6.59	11.23	5.55 ⋆ •	10.12⋆
	2	20.74	6.65	5.93⋆	10.49⋆
Carcinog.	1	30.67	4958.72	591.78•	381.89•
	2	69.78	4929.16	380.39•	242.11•
Alz_amine – PF	1	6.76	493.74	190.97•	253.41•
	2	15.58	4362.94	539.74•	519.28•
Alz_amine - NPF	1	6.76	509.79	62.31•	105.36•
	2	15.58	2710.82	124.58•	121.73•

To answer our first question we can observe from Table 1 that stochastic search can substantially improve performance compared to the original FORTE, without loss of accuracy. In the best case, Carcinogenesis over the full training set, we in fact achieve an order of magnitude improvement in performance. Revision times go down from almost 5000 seconds to less than 200 seconds. Further, the results in Table 2 suggest that the stochastic algorithms do not have worse accuracy than FORTE, and in fact actually improve accuracy over standard FORTE in a number of cases.

The second question concerns whether theory revision systems can improve predictive accuracy over the theories generated by a state-of-the art ILP system such as Aleph. Our results suggest this to be indeed the case, at least for these benchmarks. In all four cases, the revised theories significantly outperformed the theories obtained by cover removal.

Table 2. Predictive accuracy of Aleph, FORTE and stochastic revision algorithms

Dataset	Method	Aleph accuracy	Forte accuracy	SLS HC accuracy	SLS greedy accuracy
Mutag.	1	77.52	78.08	82.37 ⋆ •	78.94⋆
	2	73.08	73.63	79.74 ⋆ •	80.16 ⋆ •
Carcinog.	1	50.58	54.91	61.60 ⋆ •	59.91 ⋆ •
	2	54.94	59.20	61.84 ⋆ •	61.21 ⋆ •
Alz_amine – PF	1	62.11	63.79	68.13 ⋆ •	66.53 ⋆ •
	2	62.19	72.56	69.56⋆	66.83⋆
Alz_amine – NPF	1	62.11	62.91	68.57 ⋆ •	67.71 ⋆ •
	2	62.19	72.14	70.59⋆	68.75⋆

Our results do not show an advantage in starting from a theory obtained from half the training examples, or from using the full training-set. Notice that Aleph itself achieves similar performace in both cases.

Our results suggest that revision systems can be of interest in improving theories obtained by systems such as Aleph. Moreover, stochastic search appears as fundamental in reducing the time overhead associated with theory revision. Revising theories with FORTE can be two orders of magnitude than the initial Aleph run, and stochastic search reduces this overhead one order of magnitude in the two most difficult datasets. Even so, theory revision is still quite expensive. One way towards to achieve further improvements is to use Mode Directed Inverse Entailment search [4]. Another way is to randomize the search for revision points. We are enhancing in these ways the algorithms presented here.

We further studied the source of the speedups in more detailed experiments [6], which are not exhibited here due to space limitations. An interesting issue results from the observation that we randomize the search for revisions and the search for which antecedent to consider in a revision point. How do these contribute to the total running times? The experiments showed that randomising antecedent search introduces the main benefit, as the number of possible antecedents is is usually much larger than the number of possible revisions. On the other hand, the results showed that combining both techniques does perform better than using either techniques by itself. The results presented here consider the algorithms executing in this combined way.

6 Conclusions

We designed and evaluated a number of stochastic local search techniques for the revision of first-order theories. Our results indicate that such techniques can significantly improve the run-time and even the accuracy of a revision algorithm. Although much of the benefit comes from reducing the cost of searching new antecedents to add to a clause in a theory, as this search is quite expensive and seems to dominate revision costs, benefits are also achieved through randomizing search from revision operators.

Our current results suggest that improving antecedent generation should be a major concern in theory revision systems. One interesting future work would therefore be to constrain this space by using modes and the bottom-clause to guide the revision process (remembering that FORTE was based on FOIL).

Our results also seem to indicate that stochastic search has the potential to very significantly improve the performance of theory-revision systems, and that such improved systems may be useful in improve the theories generated by Inductive Logic Programming systems. We believe this is because theory revision systems can take a global perspective of theory, in contrast to the local approach used by the greedy cover-removal algorithms. Moreover, if Theory Revision systems can be made more efficient, this would make it more practical to revise theories given new examples. The experiments presented here suggest this may be indeed the case, and that research in this direction may be very worthwhile.

Acknowledgments

The authors are financially supported by CAPES, CNPq and Fundação para a Ciência e Tecnologia, respectively. We thank Bradley Richards and Raymond Mooney for making the FORTE system available and Kate Revoredo for useful discussions.

References

1. Bratko, I.: Refining complete hypotheses in ILP. In: Džeroski, S., Flach, P.A. (eds.) ILP 1999. LNCS (LNAI), vol. 1634, pp. 44–55. Springer, Heidelberg (1999)
2. Chisholm, M., Tadepalli, P.: Learning decision rules by randomized iterative local search. In: Proc. of the 19th ICML, pp. 75–82 (2002)
3. King, R.D., Sternberg, M.J.E., Srinivasan, A.: Relating chemical activity to structure: An examination of ILP successes. New Generation Computing 13(3-4), 411–433 (1995)
4. Muggleton, S.: Inverse entailment and Progol. New Generation Computing 13, 245–286 (1995)
5. Paes, A., Železný, F., Zaverucha, G., Page, D., Srinivasan, A.: ILP through propositionalization and stochastic k-term DNF learning. In: Muggleton, S., Otero, R., Tamaddoni-Nezhad, A. (eds.) ILP 2006. LNCS (LNAI), vol. 4455, pp. 379–393. Springer, Heidelberg (2007)
6. Paes, A., Zaverucha, G., Costa, V.S.: Further results on revising first-order theories through Stochastic Local Search. In: Technical report, Federal University of Rio de Janeiro (2007)
7. Richards, B.L., Mooney, R.J.: Automated refinement of first-order Horn-clause domain theories. Machine Learning 19(2), 95–131 (1995)
8. Rückert, U., Kramer, S.: Stochastic local search in k-term DNF learning. In: Proc. of the 20th ICML, pp. 648–655 (2003)
9. Selman, B., Kautz, H.A., Cohen, B.: Local search strategies for satisfiability testing. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 26, 521–532 (1996)
10. Selman, B., Levesque, H.J., Mitchell, D.G.: A new method for solving hard satisfiability problems. In: Proc. of the 10th AAAI, pp. 440–446 (1992)
11. Srinivasan, A.: The Aleph Manual (2001)
12. Srinivasan, A., King, R.D., Muggleton, S., Sternberg, M.J.E.: Carcinogenesis predictions using ILP. In: Džeroski, S., Lavrač, N. (eds.) ILP 1997. LNCS, vol. 1297, pp. 273–287. Springer, Heidelberg (1997)
13. Srinivasan, A., Muggleton, S., Sternberg, M.J.E., King, R.D.: Theories for mutagenicity: A study in first-order and feature-based induction. Artificial Intelligence 85(1-2), 277–299 (1996)
14. Železný, F., Srinivasan, A., Page, D.: Randomised restarted search in ILP. Machine Learning 64(1-3), 183–208 (2006)
15. Wrobel, S.: First-order theory refinement. In: De Raedt, L. (ed.) Advances in Inductive Logic Programming, pp. 14–33. IOS Press, Amsterdam (1996)

Using ILP to Construct Features for Information Extraction from Semi-structured Text

Ganesh Ramakrishnan[1], Sachindra Joshi[1], Sreeram Balakrishnan[1], and Ashwin Srinivasan[1,2]

[1] IBM India Research Laboratory, Block 1, Indian Institute of Technology, New Delhi 110016, India
{ganramkr, jsachind, srbalakr, ashwin.srinivasan}@in.ibm.com
[2] Dept. of CSE & Centre for Health Informatics, University of New Kensington, Sydney, Australia

Abstract. Machine-generated documents containing semi-structured text are rapidly forming the bulk of data being stored in an organisation. Given a feature-based representation of such data, methods like SVMs are able to construct good models for information extraction (IE). But how are the feature-definitions to be obtained in the first place? (We are referring here to the representation problem: selecting good features from the ones defined comes later.) So far, features have been defined manually or by using special-purpose programs: neither approach scaling well to handle the heterogeneity of the data or new domain-specific information. We suggest that Inductive Logic Programming (ILP) could assist in this. Specifically, we demonstrate the use of ILP to define features for seven IE tasks using two disparate sources of information. Our findings are as follows: (1) the ILP system is able to identify efficiently large numbers of good features. Typically, the time taken to identify the features is comparable to the time taken to construct the predictive model; and (2) SVM models constructed with these ILP-features are better than the best reported to date that rely heavily on hand-crafted features. For the ILP practioneer, we also present evidence supporting the claim that, for IE tasks, using an ILP system to assist in constructing an extensional representation of text data (in the form of features and their values) is better than using it to construct intensional models for the tasks (in the form of rules for information extraction).

1 Introduction

The amount of text data available in a machine-readable format is already very large: Google alone indexes more than 10 billion pages, most of which are text. This is only expected to increase, as organisations increasingly employ machines capable of generating semi-structured (XML-like) text data (for example, projections by IBM in that corporation's Global Technology Outlook for 2003 suggests that by 2010, nearly 75% of the data stored in an organisation may of this type). This trend is accompanied by a substantial industrial impetus to develop automated methods for extracting information of potential commercial interest from

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 211–224, 2008.

such data. Information Extraction (IE), normally studied under the umbrella of "text mining" ([10,6]), involves a number of tasks like: sentence segmentation [22], part of speech tagging [5], noun-phrase coreferencing [24] and named entity annotation [3]. The principal goal of IE is to extract structured information from unstructured text documents. This structured information is normally in a form that can be stored in a database: although modern relational database implementations allow the direct storage and limited querying of XML-like data, a representation using attributes (features in the machine-learning sense) remains the most popular. This makes the data amenable to not just efficient querying, but to a variety of parametric and non-parametric techniques for mining and modelling (like association-rule mining [1], naive Bayes [17], hidden Markov models [10], maximum entropy models [3], support vector machine [4] and conditional random fields [13]). These methods construct models using some subset of the features identified to represent the text.

In this paper, we are concerned with the question of how an appropriate feature representation is to be arrived at the first place. This is a step before feature-subset selection: there, once a set of features have been defined, a subset of them is sought—usually, but not always, with a view of building a good predictive model. However, obtaining definitions of the features *ab initio* is a more complex business. The "feature engineer" has to construct these from some combination of general-purpose and problem-specific information, and, if available, knowledge of how the features would be used (for example, to build a model that can discriminate accurately amongst interesting and uninteresting corporate mergers). This has meant that the task of defining the features has been one that has largely been manual, or achieved through problem-specific programs that use the knowledge sources in a pre-defined manner. It is difficult to see how these approaches could scale-up to meet the demands imposed by the scale and heterogeneity of text-based data that are being generated, or to incorporate new sources of information that become electronically available. Our interests are therefore in automated methods that can assist in this by automatic identification of interesting features. We adopt the following positions: (1) Formal logic, at least with the power embodied within logic programming languages, is adequate for representing the different kinds of information that are needed for IE; and (2) Any automatic method for identifying feature definitions that uses a logical representation has to be at least at the level of first-order logic.

We will ask the reader to take (1) as axiomatic for this paper. Some justification for (2) on the other hand, follows from the observation that feature definitions in logic are essentially functions of the form $f : X \mapsto Y$, from the set of individuals X to some more or less arbitrary Y (Y could be the Booleans, for example). Thus, if these definitions are to be identified automatically, then the underlying program has to be able to construct functions. This implies any program for feature engineering must at least be in a position to construct definitions in first-order logic (or higher, if composition of functions are needed to construct complex features; or if the representation of individuals could in turn employ functions as in [16]).

Arguably the most well-developed and general-purpose programs for constructing first-order definitions have been in the area of Inductive Logic Programming (ILP). ILP programs are normally, but not exclusively, used to learn rules (called theories in the literature) in first-order logic to classify examples. The rules employ predicates provided as background knowledge encoded in some subset of first-order logic. In this paper, we propose that ILP could be employed to attain the IE goal of converting unstructured text data to a structured form: the process we envisage is summarised in Fig. 1.

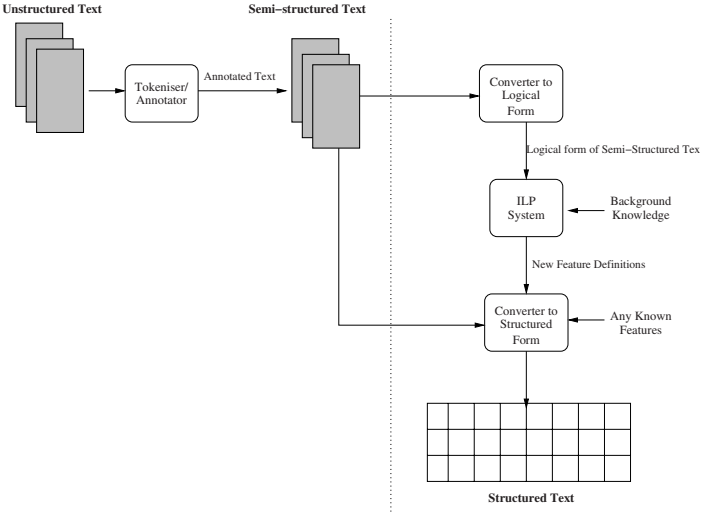


Fig. 1. A simplified view of a role for ILP in information extraction. The focus of this paper is on programs, inputs, and outputs to the right of the dotted line.

In the past, several authors have used ILP, or ILP-inspired systems for information extraction. Notable amongst these are: the work of Aitken [2] who uses ILP to construct theories for IE; Califf's work with with Rapier [7], which is inspired by bottom-up ILP systems; and the work of Roth and colleagues [23] who use restricted templates defined by "relation generating functions" to construct features for IE (their motivation for this is that general-purpose ILP methods are inflexible, making their use impossible in NLP-like domains). Our results here are intended to add these by providing evidence for the following: (1) a general-purpose ILP system can efficiently extract useful features for information extraction; and (2) feature extraction is a more effective way to use a general-purpose ILP system than the usual process of extracting rules.

Although the use of ILP as a general-purpose mechanism for identifing feature definitions has a long history (a process termed "propositionalisation": see [12] for a detailed survey), and has been shown to be one of the most effective ways to use ILP to address difficult problems (see for example: [11]), there has not been,

to the best of our knowledge, any attempt to use them for this purpose in IE. We believe that there may be two reasons for their neglect in IE. First, ILP has been perceived as being too inefficient to identify a suitably large set of features that may be needed to represent documents adequately (for example, [23] use this as their primary motivation to develop the restricted approach) . Second, it is not apparent that an automatic feature construction method could match the kinds of performance achievable by good hand-crafted features. In this paper, it is our intention to demonstrate using an established test-bed that both these concern may be unfounded. Specifically, we examine a task in information extraction concerned with the problem of extracting instances of a structured target schema from an unstructured text data. The data we use concerns corporate mergers and acquisitions, from which we intend to extract the names of the acquiring and acquired companies; and the deal amount (the actual task, described in Section 3.2 actually involves identifying seven different entities). For this task, we use a well-known ILP system to identify efficiently a large number—from a few thousand to ten thousand— good features using two quite different sources of information (Wordnet [18] and a dependancy parser). The features identified are used by a standard support vector machine (SVM) classifier to construct predictive models for the entities of interest. These models—a special kind of the type proposed by [20]—are compared against the best reported in the literature (these rely largely on hand-crafted features).

The rest of the paper is organised as follows. Section 2 describes the aspects of ILP to the extent that it relates to the feature-identification task undertaken here. Section 3 describes the empirical investigation undertaken in the paper. This includes a description of aims (Section 3.1), materials (Section 3.2), methods (Section 3.3) and results (Section 3.4). Section 4 concludes the paper.

2 Feature Definitions Using Inductive Logic Programming

Given problem-specific data and general-purpose ("background") knowledge encoded in some logical form—normally a subset of first-order logic—an ILP system attempts to construct models, also in a logical form, for the data. Implementations have been dominated by two classes of programs, corresponding somewhat to the broader division into supervised and unsupervised learning. The first class—predictive ILP—is concerned with constructing "theories" (sets of rules; or first-order logic variants of classification or regression trees) for discriminating accurately amongst two sets of examples ("positive" and "negative"). The second—descriptive ILP—has is concerned with identifying relationships that hold amongst the data and the background knowledge a view of discrimination. More details of the requirements of programs in these two categories can be found in [19]:

The task of finding the definition of features using a first-order logic representation is one that is not easily characterised as either predictive or descriptive ILP. Solutions conceptually involve two steps: (1) a feature-construction step

that identifies (within computational reason) all the features that are consistent with the constraints provided by the background knowledge. This is characteristic of a descriptive ILP program; and (2) a feature-selection step that retains some of the features based on their utility estimated using the problem-specific data. This is characteristic of a predictive ILP program. A partial specification for an ILP program that reflects this combination has been recently proposed in [26], which we follow here (we refer the reader to [21] for definitions of the logical terms used below):

- B (background knowledge) consists of a finite, possibly empty, set of clauses $= \{C_1, C_2, \ldots\}$
- E (data) consists of a finite set $E^+ \cup E^-$ where:
  - *Positive Examples*. $E^+ = \{e_1, e_2, \ldots\}$ is a non-empty set of definite clauses;
  - *Negative Examples*. $E^- = \{\overline{f_1}, \overline{f_2} \ldots\}$ is a set of Horn clauses (this may be empty)
- $\mathcal{H}$ is the set of definite clauses, constructable with predicates, functions and constants in $B \cup E$; $\mathcal{F}$ the set of features constructable using a set of individuals; and $\tau : \mathcal{H} \mapsto \mathcal{F}$ a function that maps a definite clause $h \in \mathcal{H}$ to a feature $f \in \mathcal{F}$.
- $F = \{f_1, f_2, \ldots\} \in \mathcal{F}$, the output of the algorithm given B and E is acceptable for any set $H = \{h_1, h_2, \ldots\} \in \mathcal{H}$ if the following conditions are met:
  - *Posterior Sufficiency*. $B \cup \{h_i\} \models e_1 \vee e_2 \vee \ldots$, where $\{e_1, e_2, \ldots\} \subseteq E^+$
  - $f_i = \tau(h_i)$

The reader would have noted that Posterior Sufficiency requires features constructed here use clauses that entail, with B, at least one positive example: this is not necessarily satisfied by programs that simply seek to identify all features constructable with predicates, functions and constants in $B \cup E$. The features returned by such programs will be a superset of the features F. Here, we seek instead to constrain the set F further to return a subset F' of F that accounts for a notion of utility using some predicate *Good* (effectively, being the same notion of "interestingness" used in the data-mining literature):

- *Utility*. $F' = \{f : f \in F,\ and\ Good(f, B, E) = TRUE\}$

For the rest, we follow [26] and refer the reader to that paper for details of $\mathcal{F}$, $\mathcal{H}$ and τ. We view the definition of *Good* as problem-specific, and details for the empirical study here are provided in Section 3.3. A program that satisfies these requirements constructs the definition of a feature in the following manner. First, a set of clauses H is identified using the examples and background knowledge. Each clause is of the form $head \leftarrow body$, where *head* is a literal and *body* a conjunction of literals; and entails at least one positive example, given the

background knowledge B. Next, each clause h_i in H is converted into a boolean feature f_i that takes the value 1 (or 0) for any individual for which the body of the clause true (if the body is false).[1] Thus, the set of clauses H gives rise to a boolean vector for each individual in the set of examples. An example in the context of information extraction is shown in Fig. 2, in which the task is to assign "roles" to individuals. The problem concerns corporate mergers and acquisitions, and individuals are assigned roles like "purchaser", "purchased", "deal amount" and so on. In the example in Fig. 2, individuals are identified by the triple $\langle d, s, l \rangle$, where d denotes a document, s a sentence in d, and l the location of a segment in d.

Clause:

$$\begin{aligned}&\forall d, s, l(Has\_role(d, s, l, purchaser) \leftarrow \\ &\quad Has\_annotation(d, s, l, \text{organisation}) \wedge \\ &\quad After(l, l1) \wedge \\ &\quad Has\_hyp\_sense(d, s, l1, 02\text{incrs0incrm0}))\end{aligned}$$

Feature:

$$f(d, s, l) = \begin{cases} \begin{array}{l} Has\_annotation(d, s, l, \text{organisation}) \wedge \\ After(l, l1) \wedge \\ Has\_hyp\_sense(d, s, l1, 02\text{incrs0incrm0}) \\ \quad = TRUE \end{array} & 1 \\ 0 \text{ otherwise} & \end{cases}$$

Fig. 2. Example of a boolean feature constructed from a clause. The clause assigns the role 'purchaser' to any individual (denoted by specific values assigned to variables d, s and l) that satisfies the conditions in the body of the clause. The meanings of the predicate symbols $Has\_annotation$, $After$, and Has_hyp_sense are explained in Section 3.

3 Experimental Evaluation

3.1 Aims

We intend to investigate the use of an ILP system for automatic feature construction in information extraction. Specifically, using a benchmark dataset, our primary goals are to examine: (a) Whether the ILP system is able to identify features efficiently using multiple sources of background knowledge[2] and (b) Whether the features identified by an ILP system are "good", measured by

[1] The body forms the definition of a "context predicate" in the terminology of [22].

[2] In principle, it is evident that an ILP capable of using one source of background information should be able to use multiple sources as well. However, the sub-optimal nature of most implementations has meant that this is not necessarily the case in practice.

comparing the predictive models that are constructed using the features—we will call the approach "ILP-assisted"—against the best reported models (these use a combination of hand-crafted and simple automatically constructed features).

3.2 Materials

Data. We use data contained in the "Corporate Acquisition Events" corpus described in [14]. This is a collection of 600 news articles describing acquisition events taken from the Reuters dataset. News articles are tagged to identify fields related to acquisition events. These fields include 'purchaser' , 'acquired', and 'seller' companies along with their abbreviated names ('purchabr', 'acqabr' and 'sellerabr') Some news articles also mention the field 'deal amount'. Together, these seven fields define the set of target elements for information extraction task: we will refer to these fields as "roles" in the rest of the document. In Table 1, we summarize this information.

Table 1. Examples in the Corporate Acquisitions Events corpus

Role	Number of Examples
acquired	651
acqabr	1494
purchaser	594
purchabr	1347
seller	707
sellerabr	458
deal amount	206
Total	5457

Each unstructured text document is first converted to a semi-structured form using a tokeniser, followed by an annotator. The output of each of these are then converted automatically into a logical form, which is then part of the data provided to the ILP system. The details are as follows.

Each unstructured text document contains one or more sentences, each of which can have several tokens. Tokens are individual words, or groups of words which have a unique identifier within a sentence in a document. Tokens are then tagged using a high-recall named entity annotator. We use a rule-based named entity annotator developed in-house, that produces four different annotations: 'currency', 'date', 'location' and 'organisation'.

We convert the output of a tokeniser into a logical form that encodes the location of tokens. This is done using the predicate Has_token, resulting in facts of the form *Has_token (doc_id, sentence_id, token_id, t)*. This states that at there is a token t at location *token_id* within *sentence_id* in document *doc_id*. For the dataset here, there are approximately $70,000$ statements of this form. In a similar manner, the results of the annotator are encoded by facts of the

form *Has_annotation(doc_id, sentence_id, token_id, a)*. There are approximately 10,000 statements of this form.

Examples of roles identified in the text are encoded using the predicate *Has_role*. The result is a set of facts of the form *Has_role (doc_id, sentence_id, token_id, r)*. Corresponding to the entries in Table 1, there are 651 facts with role *acquired*, 1494 with *acqabr* and so on. The entire corpus used is thus represented in a logical form by approximately 85,000 facts.

We further generate "negative examples" for tokens at a position (that is, at a location in a sentence within a document) in the following manner. First, roles not assigned to a token at a position are each taken to constitute a negative example for the token at that position. Second, a token at a position that has been annotated by the named entity annotator, but does not have a role assigned to it at that position is marked as having none of the possible seven roles at that position. This results in approximately in a further 71,000 facts. The entire dataset is thus represented by about 156,000 facts.

Background Knowledge. Background knowledge for the ILP system consists of logical encodings of the following two sources of information:

Semantic Lexicon. Natural languages provide a rich set of expressions allowing several different ways of expressing the same fact. As an example, the expression 'company A purchased company B recently' and the expression 'company A recently acquired company B' mean the same thing. We use WordNet to help address some of this problem. WordNet is a semantic lexicon for the English language. It groups English words into sets of synonyms called 'synsets' and records the various semantic relations between these synonym sets. We use the hypernym relations within Wordnet to compute the hypersense of tokens. This requires the following computation. First, the synsets corresponding to a token are obtained. Hypernyms of each synset are hyper senses for the token. Further hyper senses can obtained recursively from hypernyms of the synsets, their hypernyms and so on. We restrict ourselves to two levels of such recursion. The result is encoded using predicates of the form: *Has_hyp_sense(doc_id, sentence_id, token_id, s)*. While any modern ILP system can compute these facts "on-the-fly", we pre-compute them for efficiency. This results in approximately 531,000 facts.

Dependency Parser. We use MINIPAR [15] to obtain dependency relationships in a sentence. A dependency relationship is an asymmetric relationship between a token called *head* or parent and another token called *modifier* or dependent. A token in a sentence may have several modifiers, however, it can modify only a single word. The root of a dependency tree does not modify any word and is called the head of the sentence. A MINIPAR 'relationship' is a label assigned to a dependency relationship between a pair of tokens in a sentence. Some of the examples relationships used in MINIPAR are 'subj' (subject), 'adjn' (adjunct), 'cmpl' (complement), and 'obj' (object). We generate a set of facts of the following form using the output generated by MINIPAR: *Links(doc_id, sentence_id, link_type, token_id_1, token_id_2)*. The

argument *Link_type* can take values from the set of relations used by MINIPAR. The arguments *token_id_1* and *token_id_2* correspond to the modifier and the header word respectively. Again, all these facts could be computed on-th-fly, but are precomputed here. There are approximately 76,000 facts of this form.

In addition, we also provide utility predicates *After* and *Before* that allow the ILP program to access locations in the neighbourhood of any given location in a sentence. The background information is thus represented by about 600,000 facts.

3.3 Method

Our method for estimating the performance of ILP-assisted approach is straightforward:

Repeat N times:

1. Randomly split the data into training (Tr) and test (Te) data;
2. Obtain a set of no more than k features using the ILP program using background knowledge B;
3. Construct a classificatory model for predicting the roles in Fig. 1 using a standard classification technique equipped with the features identified in Step 2 and the data in Tr;
4. Record the performance of the model obtained in Step 3 on the data in Te;

Estimate the predictive performance of ILP-assisted approach by the mean values of accuracies obtained in Step 4.
Compare the performance of the models with ILP-assisted approach against performance of the best models reported in the literature.

The following details are relevant:

(a) We take N to be 5 for our experiments;
(b) k is restricted to 20,000 for our experiments;
(c) We use the ILP system Aleph (Version 5) [27] for all experiments. This program has a procedure that constructs features using a non-greedy set covering approach (we refer the reader to the Aleph manual for details). The procedure ensures that features are constructed from clauses that satisfy the Posterior Sufficiency requirement in Section 2. The classificatory model in Step 3 is obtained using a linear SVM: the specific implementation used is the one provided in the WEKA toolbox called SMO.[3] It can be shown that the resulting models are a special case of the SVILP models proposed in [20]. We compare the performance of this model against those constructed by SRV [9], HMM [10] and Elie [8].

[3] `http://www.cs.waikato.ac.nz/ ml/weka/`

(d) Satisfying the Utility requirement requires the definition of the predicate *Good*. Here, *Good* is $TRUE$ for clauses (and, by implication, features found by the ILP system) that entail at least 5 positive examples and have a precision of at least 0.6. These numbers are arbitrary, but do not seem to affect the results greatly; The implementation we use is also able to ensure that this utility requirement is met during its search for features: this is more efficient than first finding all features and then removing those that fail to meet the criterion of minimum utility. All other settings for Aleph are at their default values for feature construction.

(e) Although the utility predicates $Before$ and $After$ allow access to any number of locations within a sentence, for experiments here we restrict the neighbourhood to no more than 5 locations on either side of a token.

(f) Performance will be measured using the following standard statistics: precision (P), recall (R), and F1. These are defined as follows: $P = TP/PredPos$, where TP is the true positives and $PredPos$ are the numbers of examples predicted as positive; $R = TP/ActPos$, where $ActPos$ are the numbers of examples that are actually positive; and $F1 = 2PR/(P + R)$.

(g) A quantitative comparison of the performance of the ILP-assisted approach are only possible against Elie (predictions for SRV and HMM are only available for 3 of the 7 roles, which makes any quantitative statement unreliable). For this, we use the Wilcoxon signed-rank test [25]. The test is a non-parametric test of the null hypothesis that there is no significant difference between the median F1 performance of a pair of algorithms.

3.4 Results

Figure 3 shows the performance of the classificatory model obtained using ILP-assisted approach; and Fig. 4 shows the comparative performance of this against the best reports available. On balance, the signed-rank test suggests that there is some evidence in favour of the ILP-assisted approach over Elie, although the difference is not significant (the sum of signed ranks is 14: the critical value for a

Role	**Performance**		
	P	R	F1
acquired	51.7	35.2	41.8
acqabr	46.0	39.8	42.6
purchaser	54.7	39.0	45.4
purchabr	42.3	30.8	35.4
seller	52.2	52.1	51.5
sellerabr	25.4	19.5	21.7
dlramt	60.9	47.3	53.0

Fig. 3. Estimates of performance of the ILP-assisted classifier. P, R denote precision and recall. F1 is the harmonic mean of P and R.

significant difference is 22). Numbers are too small to make a reliable statement about other comparisons: the odds seem to favour the ILP-assisted approach over either SRV or HMM. These results suggest that the ILP-assisted approach is at least as good as the others tabulated.

Comparative assessments of this nature, while valuable, are unrepresentative of how the ILP-assisted approach is intended to be used in practice. As shown in Fig. 1, we envisage the features identified by the ILP algorithm to augment any existing features. It is therefore of more interest to see how a methods like Elie perform when provided additionally, with the ILP features; or conversely, providing SMO (the SVM implementation used here to build models with the ILP features) additionally with the features used by methods like Elie, HMM and SRV. Unfortunately, we are not able to perform either of these experiments: executable implementations of Elie, SRV and HMM are not available, and we are unable to re-construct the definitions of their features from the descriptions provided in the literature. A surrogate experiment is however possible, that gives some insight into what may be achievable in practice. We are able to examine the construction of features by the ILP system without the semantic lexicon and the dependency parser. This amounts to the ILP system only having access to the tokeniser, annotator, and the utility predicates $Before$ and $After$. In such circumstances, the features constructed will be regular expressions, referring to the locations of tokens, annotations and gaps. Taking this as a kind of baseline for either hand-crafted or features that can be constructed (this does not need a general-purpose ILP system), we are in a position to examine the change in performance effected by the ILP system as it identifies features by incorporating background information. From Fig. 5, it is evident that models that use ILP features constructed with additional background knowledge do perform better than the baseline. This suggests that the ability of the ILP system to use background

Role	SRV	HMM	Elie	ILP-assisted
acquired	34.3	30.9	42.0	41.8
acqabr	35.1	40.1	40.0	**42.6**
purchaser	42.9	**48.1**	47.0	45.4
purchabr	−−	−−	29.0	**35.4**
seller	−−	−−	15.0	**51.5**
sellerabr	−−	−−	14.0	**21.7**
dlramt	−−	−−	**59.0**	53.0

Fig. 4. Comparative performance of the ILP-asssisted approach. Results for three roles are not reported by SRV and HMM.

Role	**B**	**B+L**	**B+P**	**B+L+P**
acquired	34.5	40.2	41.5	41.8
acqabr	34.5	39.8	42.6	42.6
purchaser	42.5	42.8	45.2	45.4
purchabr	25.0	30.7	29.4	35.4
seller	44.3	48.8	50.6	51.5
sellerabr	19.8	23.2	19.3	21.7
dlramt	49.2	50.3	47.1	53.0

Fig. 5. Comparative F1 values for models constructed by SMO. "B" refers to models constructed using baseline features that are regular expressions on tokens and annotations. "L" refers to semantic lexicon features that use information provided by Wordnet, and "P" refers to features that use information provided by the dependency parser MINIPAR.

information does translate into finding features that can improve predictive performance. If the results in Fig. 5 are representative, then augmenting the baseline features with ILP features that use all the background information yields improvements in the F1 value of 5.5. This can be taken as some indication of the improvements that may be achievable by augmenting SMO with the features used by Elie, SRV or HMM.

It is also of some interest to consider for the specific IE tasks considered here whether it is better to use the ILP-assisted approach proposed; or to use ILP to construct models that directly predict the roles of individuals in documents (we will call this "ILP-models"). Evidence that we have suggests the former is better: see Fig. 6.

Method	Performance
ILP-assisted	39.8 ±0.9
ILP-models	35.2 ±1.5

Fig. 6. Average F1 values for models using the ILP-assisted approach and ILP models that predict roles directly. The averages are weighted averages that account for the proportions of examples for each role.

We have not commented so far on the efficiency of constructing features using ILP. Our estimates suggest that in all cases, the time for feature construction is comparable to the time taken for model construction with the SVM. Some caveats are needed. Model construction with an SVM is faster than feature construction (but no more than about 5 times) when: (a) the number of features are small (say about 2000 or so); and (b) the number of classes are small. As either of these are increased, model construction time was observed to be significantly greater than the time taken to construct features.

4 Concluding Remarks

In this paper, we have investigated the use of ILP to assist in the task of identifying features for converting text data into a structured form. Our results suggest that an ILP system can effectively amalgamate information from disparate sources to identify features that can then be used to build good predictive models for specific IE tasks. The promise of adopting this route rests on two points: (1) ILP provides a general-purpose setting for feature-construction that uses a substantially rich subset of first-order logic for representation, and makes explicit provision for incorporating domain-specific and general-purpose background information; and (2) features constructed in this manner augment, not replace, those already known to be effective. In principle, the output of any good model builder should not get any worse.

While the experimental results are sufficient to demonstrate the feasibility and the value obtained using an ILP-assisted approach, there are three ways

in which they could be extended immediately. First, similar positive results on other IE tasks would clearly establish the utility of the approach further. Second, we have used a very general-purpose ILP system in our experiments. ILP implementations that are specifically designed for feature-identification now exist (these do not lose any of the other generality of ILP). These would provide more efficient identification of features (and possibly even better ones). Third, we intend using other knowledge sources such as VerbNet in future work.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proc. 20th Int. Conf. Very Large Data Bases, VLDB, pp. 12–15 (1994)
2. Aitken, J.S.: Learning Information Extraction Rules: An Inductive Logic Programming approach. In: Proceedings of the 15th European Conference on Artificial Intelligence, pp. 355–359 (2002), http://citeseer.ist.psu.edu/586553.html
3. Borthwick, A.: A maximum entropy approach to named entity recognition. PhD thesis (1999)
4. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: 5th Annual ACM Workshop on COLT, pp. 144–152 (1992)
5. Brants, T.: Tnt: a statistical part-of-speech tagger. In: Proceedings of the sixth conference on Applied natural language processing (2000)
6. Bunescu, R., Mooney, R.J.: Relational markov networks for collective information extraction. In: Proceedings of the ICML-2004 Workshop on Statistical Relational Learning and its Connections to Other Fields (2004)
7. Califf, M.E., Mooney, R.J.: Relational learning of pattern-match rules for information extraction. In: Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing (1998)
8. Finn, A., Kushmerick, N.: Multi-level boundary classification for information extraction. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) ECML 2004. LNCS (LNAI), vol. 3201, pp. 111–122. Springer, Heidelberg (2004)
9. Freitag, D.: Toward general-purpose learning for information extraction. In: Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics (1998)
10. Freitag, D., McCallum, A.K.: Information extraction with hmms and shrinkage. In: Proceedings of the AAAI 1999 Workshop on Machine Learning for Informatino Extraction (1999)
11. King, R.D., Srinivasan, A., DeHaspe, L.: WARMR: A Data Mining Tool for Chemical Data. Computer Aided Molecular Design 15, 173–181 (2001)
12. Kramer, S., Lavra, N., Flach, P.: Propositionalization approaches to relational data mining. Springer, New York (2000)
13. Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proc. 18th International Conf. on Machine Learning, pp. 282–289. Morgan Kaufmann, San Francisco (2001)
14. Lewis, D.D.: Representation and learning in information retrieval. PhD thesis (1992)
15. Lin, D.: Dependency-based evaluation of minipar. In: Workshop on the Evaluation of Parsing Systems (1998)
16. Lloyd, J.W.: Logic for learning: Learning comprehensible theories from structured data. Cognitive Technologies Series. Springer, Heidelberg (2003)

17. McCallum, A., Nigam, K.: A comparison of event models for naive bayes text classification. In: AAAI 1998 Workshop on Learning for Text Categorization (1998)
18. Miller, G.: Wordnet: A lexical database for english. Commun. ACM 38(11) (1995)
19. Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. Journal of Logic Programming 19(20), 629–679 (1994)
20. Muggleton, S.H., Lodhi, H., Amini, A., Sternberg, M.J.E.: Support Vector Inductive Logic Programming. In: Hoffmann, A., Motoda, H., Scheffer, T. (eds.) DS 2005. LNCS (LNAI), vol. 3735, pp. 163–175. Springer, Heidelberg (2005)
21. Nienhuys-Cheng, S., de Wolf, R.: Foundations of Inductive Logic Programming. Springer, Berlin (1997)
22. Reynar, J.C., Ratnaparkhi, A.: A maximum entropy approach to identifying sentence boundaries. In: ANLP, pp. 16–19 (1997)
23. Roth, D., Yih, W.t.: Relational learning via propositional algorithms: An information extraction case study. In: IJCAI, pp. 1257–1263 (2001)
24. Sang, E.F.T.K., Daelemans, W., Déjean, H., Koeling, R., Krymolowski, Y., Punyakanok, V., Roth, D.: Applying system combination to base noun phrase identification. In: COLING, pp. 857–863 (2000)
25. Siegel, S., Castellan Jr, N.J.: Nonparametric Statistics for The Behavioral Sciences. McGraw-Hill, New York (1956)
26. Specia, L., Srinivasan, A., Ramakrishnan, G., Nunes, M.G.V.: Word sense disambiguation using ilp. In: 16th International Conference on Inductive Logic Programming (2006)
27. Srinivasan, A.: The Aleph Manual (1999), http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/

Mode-Directed Inverse Entailment for Full Clausal Theories

Oliver Ray[1] and Katsumi Inoue[2]

[1] University of Bristol, United Kingdom
oray@cs.bris.ac.uk
[2] National Institute of Informatics, Japan
ki@nii.ac.jp

Abstract. Mode declarations are a successful form of language bias in explanatory ILP. But, while they are heavily used in Horn systems, they have yet to be similarly exploited in more expressive clausal settings. This paper presents a mode-directed ILP procedure for full clausal logic. It employs a first-order inference engine to abductively and inductively explain a set of examples with respect to a background theory. Each stage of hypothesis formation is guided by mode declarations using a generalisation of efficient Horn clause techniques for inverting entailment. Our approach exploits language bias more effectively than previous non-Horn ILP methods and avoids the need for interactive user assistance.

1 Introduction

Mode declarations [7] are an important form of language bias in explanatory ILP. They are a convenient way to define the hypothesis space searched by prominent systems like Progol [7] and Aleph [14]. These systems use a type of Mode Directed Inverse Entailment (MDIE) [7] to construct and generalise a clause, called a Bottom Set [7], which bounds an otherwise intractable search space by acting as a syntactic and semantic 'bridge' between examples and hypotheses. But, while the practical utility of MDIE has been convincingly shown in Horn clause applications of ILP, so far, no methods have been studied for lifting MDIE to the more expressive setting of full clausal logic. This work aims to show the feasibility and necessity of such a generalisation and to thereby enable the representation and discovery of indefinite and disjunctive knowledge in explanatory ILP.

This paper presents a mode directed proof procedure for full clausal ILP. It uses a first-order inference engine called SOLAR [9] to implement a full clausal extension of an MDIE approach called HAIL [11]. HAIL combines abductive and inductive reasoning to construct and generalise a multi-clause variant of the Bottom Set called a Kernel Set [11]. SOLAR is an efficient tool for computing the deductive consequences of a given theory that satisfy a vocabulary known as a Production Field [2]. We use SOLAR to provide a full clausal realisation of HAIL, called fc-HAIL, by lifting the principles of MDIE from Horn clauses to general clauses. By exploiting language bias more effectively than previous

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 225–238, 2008.

non-Horn explanatory ILP methods, fc-HAIL avoids the need for interactive user assistance. The utility of fc-HAIL is illustrated by a case study in fluid modelling.

The paper is structured as follows: Section 2 reviews the relevant background material; Sections 3 and 4 present our case study and the fc-HAIL approach; Section 5 compares fc-HAIL with related work; and Section 6 concludes.

2 Background

2.1 Notation and Terminology

This paper assumes a first-order language $\mathcal{L}$ with connectives $\wedge$, $\vee$, $\neg$, $\leftarrow$, $\rightarrow$, $\leftrightarrow$, logical constants $\top$, $\bot$, and a classical entailment relation $\models$. A literal L is either an atom A or its negation $\neg A$. The complement of L, denoted $\overline{L}$, is defined as $\neg A$ (resp. A) if $L = A$ (resp. $\neg A$). A clause C is an (ordered) disjunction of literals $L_1 \vee \ldots \vee L_m$. For convenience, we write $L_i \in C$ to denote that L_i is among the disjuncts of C, and we write $D \subseteq C$ to denote that every disjunct of D is also a disjunct of C. The empty clause is denoted $\square$. All variables are implicitly universally quantified at the front of the clause. A clause is Horn iff it has at most one positive literal and is full otherwise. A clause D subsumes C, written $D \geq C$, iff there exists a substitution θ such that $D\theta \subseteq C$. Moreover, D is said to properly subsume C iff $D \geq C$ but $C \not\geq D$. A theory T is an (implicitly conjoined) set of of clauses. A theory S subsumes T, written $S \geq T$, iff each clause in T is subsumed by at least one clause in S. We are concerned with the following concrete task of explanatory ILP [7]: given theories B, E^+ and E^-, find a theory H such that $B \wedge H \models E^+$ and $B \wedge H \wedge E^- \not\models \bot$. We will also require H to satisfy some additional syntactic constraints, specified by a set M of mode declarations, as defined below.

2.2 Mode Declarations

Mode declarations are a well known form of language bias in ILP that are central to this paper. As defined in [7], a mode declaration m is either a head declaration $modeh(r, l)$ or a body declaration $modeb(r, s)$ where r is an integer (the recall) and s is a ground literal (the scheme) which can contain the place-marker terms, $\#$, $+$ and $-$. Each scheme can be regarded as a 'template' with the place-markers standing for constants, input variables and output variables, respectively. The distinction between input and output variables is given by the restriction that any input variable in a body literal must be an input variable in the head or an output variable in some preceding body literal [7].

In this way, a set M of mode declarations is associated with a set of clauses $\mathcal{L}_M$, called the language of M, such that $C = L_1 \vee L_2 \vee \ldots \vee L_n \in \mathcal{L}_M$ iff L_1 (resp. each L_i for some $1 < i \leq n$) is obtained from some head (resp. body) declaration in M by replacing all $\#$ place-markers with constants and by replacing all $+$ (resp. $-$) place-markers with input (resp. output) variables. The schema of a mode declaration m is denoted $schema(m)$ and defined as the

literal obtained from the scheme of m by replacing all place-markers by distinct variables. As explained in [7], the recall of a mode declaration is used to bound the number of times the scheme can be used (with the same sequence of input variables). An arbitrary recall is often denoted by an asterix $*$. In general, each place-marker can be optionally annotated by a predicate. which restricts the type of terms that may legally replace it, but, for simplicity, we will not consider type predicates in this paper.

Traditionally, mode declarations are used to specify Horn hypotheses, where the first (i.e., head) literal of the clause is always positive and the remaining (i.e., body) literals are all (implicitly) negative. The idea is to build hypothesis clauses incrementally from left to right starting with the head literal and then adding successive body literals. In the full clausal case, we will use mode declarations in exactly the same way. The only difference is that the head literal can be positive or negative and so can the body literals. But, to avoid any potential confusion, we will hereafter use the notation *modef* (i.e., mode first) instead of *modeh*, and *moder* (i.e., mode rest) instead of *modeb*, to emphasise that we are referring to the first literal and the remaining literals in a clause.

Example 1. If $M = \{modef(*, \neg p(+,+)), moder(*, q(+,-)), moder(*, r(+,+))\}$ then $\neg p(X,Y) \vee q(Y,Z) \vee r(Z,Y) \in \mathcal{L}_M$ but $\neg p(a,Y) \vee q(Z,Y) \vee q(Z,Y) \notin \mathcal{L}_M$ (as the first literal has a constant instead of a variable, the second has an output variable instead of an input variable, and the third has the wrong sign).

2.3 MDIE (Mode Directed Inverse Entailment)

MDIE [7] is a mode-directed ILP approach (used by several prominent systems such as Progol and Aleph) based on the semantics of Bottom Generalisation (BG) [7]. The principle is to explain an example clause E relative to a background theory B by constructing and generalising an intermediate clause called a Bottom Set [7] of B and E. Intuitively, the Bottom Set provides a syntactic and semantic bridge between the example and hypothesis and it serves to bound a hypothesis space that would otherwise be intractable to search. The main formal definitions and results, recalled from [7], are summarised below:

- a Bottom Set of B and E is a ground clause $L_1 \vee \ldots \vee L_n$ such that $B \wedge \neg E\sigma \models \neg L_i$ for all $1 \leq i \leq n$ and for some Skolemising substitution σ
- a (Skolem-free) clause H is said to be computed by BG from B and E iff there exists a Bottom Set C of B and E such that $H \geq C$
- for any clause H computed by BG from B and E it holds that $B \wedge H \models E$

While BG is defined for full clauses, MDIE is restricted to the Horn case. MDIE uses a mode-directed BG procedure within an ILP cover loop that selects seed examples E from a set E^+ of positive examples and ensures hypotheses H are consistent with a set E^- of negative examples. It uses a set M of mode declarations to heavily constrain the BG process. MDIE was first defined for Observation Predicate Learning (OPL) [8], but was later extended to the non-OPL

setting using an approach called Theory Completion by Inverse Entailment (TCIE) [8]. This is a three phase method where, for each seed example E:

- a Bottom Set head literal is computed using a contrapositive reasoning method to find those head declaration instances that entail $E\sigma$
- some Bottom Set body atoms are computed using a query driven saturation process to find those body declaration instances that are entailed by $\neg E\sigma$
- a hypothesis is computed by a lattice based generalisation procedure using a top down search of the subsumption lattice bounded by the Bottom Set. A compression heuristic is used to search for consistent hypotheses that contain few literals and cover many positive examples

While it is one of the most successful ILP methods to date, MDIE has two key limitations: it can only infer one clause for each seed example and it only applies to Horn clauses. As argued in Section 3, these restrictions can limit the effectiveness of MDIE in practice. The first issue has been addressed in an extension of MDIE called HAIL [11]. This paper goes on to tackle the second issue by proposing a full clausal realisation of HAIL based on a first-order inference engine called SOLAR [9]. HAIL and SOLAR are now briefly summarised.

2.4 HAIL (Hybrid Abductive Inductive Learning)

HAIL [11] is a mode-directed approach for realising the semantics of Kernel Set Subsumption (KSS) [12], which is based on a multi clause extension of the Bottom Set, called a Kernel Set [12]. KSS can be seen as a tractable extension of BG which is necessary in some applications. As explained in [12]:

- a Kernel Set of B and E is a ground theory $K = \bigcup_{i=1}^{n} L_0^i \vee L_1^i \vee \ldots \vee L_{m_i}^i$ such that $B \wedge (L_0^1 \wedge \ldots \wedge L_0^n) \models E\sigma$ and $B \wedge L_j^i \models E\sigma$ for all $1 \leq i \leq n$ and $1 \leq j \leq m_i$ for some Skolemising substitution σ
- a (Skolem-free) theory H is said to be computed by KSS from B and E iff there exists a Kernel Set K of B and E such that $H \geq K$
- for any theory H computed by KSS from B and E it holds that $B \wedge H \models E$

The Kernel Set plays an identical role to the Bottom Set by serving as an intermediate ground hypothesis that bounds the search space of an anti-subsumption generaliser. Moreover, HAIL is based on a three phase methodology which is directly analogous to MDIE in that:

- the contrapositive reasoning method of MDIE is replaced by a more general abductive procedure which can return solutions with more than one atom
- the query driven MDIE saturation process is applied, in turn, to each atom abduced in the previous step
- the MDIE compression-based search procedure is used to greedily generalise, in turn, each Kernel Set clause constructed above

By integrating abduction and induction in this way, HAIL overcomes some practical limitations of MDIE, such as an incompleteness of its contrapositive procedure and its inability to infer more than one clause for each example [10].

2.5 SOLAR (SOL Resolution for Advanced Reasoning)

SOLAR is a proof procedure for clausal consequence finding [2]. It is based on Skip Ordered Linear (SOL) resolution [2], which can be seen as extending the Model Elimination calculus [6] with a rule for 'skipping' or 'assuming' literals. Since the closure of a theory is often infinite, SOLAR computes the so-called characteristic clauses [2], which are subsume-minimal consequences that satisfy a form of language bias known as a production field [2].

A production field P is a pair $P = \langle Lits, Cond \rangle$ with a set of literals $Lits$ and a condition $Cond$. The language $\mathcal{L}_P$ of P is the set of clauses whose literals are instances of $Lits$ and which satisfy $Cond$. In practice, $Cond$ simply constrains certain properties of the clause such as its *length* and *depth*. If $Cond$ is empty, we write $P = \langle Lits \rangle$. If $Th(T)$ is the deductive closure of a theory T, and μT is the set of clauses in T not properly subsumed by another clause in T, then the characteristic clauses of T wrt P are defined as $Carc(T, P) = \mu(Th(T) \cap \mathcal{L}_P)$.

As explained in [2] and [4] SOL deductions are formally defined using the notion of a structured clause, which is a pair $\langle A, B \rangle$ consisting of two clauses A and B, where the latter may contain so-called framed literals of the form $\boxed{L}$ denoting previously resolved literals. Definition 1 is now recalled from [4].

Definition 1 (SOL Deduction). *Let T be a theory, S be a clause, and P be a production field. An SOL deduction of S from T and P (of length n) is a sequence of structured clauses $D_0, \ldots, D_n$ satisfying rules 1-6 below.*

1. *$D_0 = \langle \Box, C \rangle$ for some clause $C \in T$.*
2. *$D_n = \langle S, \Box \rangle$.*
3. *For each $D_i = \langle A_i, B_i \rangle$ clause $A_i \cup B_i$ is not a tautology.*
4. *For each $D_i = \langle A_i, B_i \rangle$ clause B_i is not subsumed by any B_j with the empty substitution, where $D_j = \langle A_j, B_j \rangle$ is a previous structured clause with $j < i$.*
5. *For each $D_i = \langle A_i, B_i \rangle$ clause A_i belongs to P.*
6. *$D_{i+1} = \langle A_{i+1}, B_{i+1} \rangle$ is obtained from $D_i = \langle A_i, B_i \rangle$ as follows:*
   (a) *let L be the left-most literal of B_i. Then A_{i+1} and R_{i+1} are obtained by applying one of the rules:*
      i. *Skip: if $A_i \cup \{L\}$ belongs to P, then $A_{i+1} = A_i \cup \{L\}$ and R_{i+1} is the clause obtained by removing L from B_i.*
      ii. *Resolve: if there is a clause E_i from $T \cup \{C\}$ such that $\neg K \in E_i$ and L and K have a most general unifier θ, then $A_{i+1} = A_i\theta$ and R_{i+1} is the clause obtained by concatenating $E_i\theta$ and $B_i\theta$, framing $L\theta$, and removing $\neg K\theta$.*
      iii. *Factoring: if A_i or B_i contains an unframed literal K such that L and K have a most general unifier θ, then $A_{i+1} = A_i\theta$ and R_{i+1} is obtained from $B_i\theta$ by deleting $L\theta$.*
      iv. *Reduction: B_i contains a framed literal $\boxed{\neg K}$, and L and K have a most general unifier θ, then $A_{i+1} = A_i\theta$ and R_{i+1} is obtained from $B_i\theta$ by deleting $L\theta$.*
   (b) *B_{i+1} is obtained from R_{i+1} by deleting every framed literal not preceded by an unframed literal in the remainder (truncation).*

The utility of SOL lies in the fact that many tasks such as abduction and query answering [13,4] can be reduced to the computation of characteristic clauses. Moreover, if $\mathcal{L}_P$ is closed under the inclusion of subsuming clauses, then P is said to be stable [2] and SOL can compute clauses in $Carc(T, P)$ under some efficient pruning strategies (such as regularity for skipped literals, order preserving reduction, lemma matching, and local failure caching [5]). These strategies are all incorporated into the implementation of SOLAR used in this paper, which we will use to provide a full clausal implementation of HAIL.

3 Motivating Example: Fluid Modelling

One benefit of non-Horn ILP is that domain knowledge can be easily formalised in first-order logic and automatically translated into clausal form. This simplifies the knowledge engineering process by affording a more direct and less error prone formulation of prior knowledge. Another benefit is that non-Horn ILP enables the exploitation and discovery of disjunctive and indefinite concepts, which may be useful in some applications. We illustrate these ideas in Example 2 below.

Example 2. This example concerns the flow of a liquid under gravity through a vertical system of pipework consisting of valves and branches. As illustrated in Figure 1, each valve has two points of connection, a top and a bottom, and it has a state which can be open or not open. Intuitively, water will flow from the top of the valve to its bottom if the valve is open (e.g. v1), but not otherwise (e.g. v2). By contrast, branches have three points of connection, a top, a bottom and a side. Ordinarily, water will flow from the top of a branch to its bottom (e.g. b2), but will be forced out of the side if it cannot flow out of the bottom (e.g. b1). A system of valves and branches is constructed by connecting the top of a branch or valve to the bottom or side of another branch or valve. Any unconnected points are designated inlets or outlets.

To model the behaviour of a fluid flowing through a system of pipework, it is convenient to introduce the notions of source and sink. Intuitively, a source is a point into which fluid will flow from an inlet, while a sink is a point from which fluid would discharge into an outlet.[1] A formal model of these concepts is given by the domain theory D in Figure 2. It shows 12 first-order implications (which are implicitly conjoined and universally quantified at the front). The first 2 implications formalise the behaviour of valves. They say the bottom of a valve is a source iff the top is a source and the valve is open; and vice versa. The next 4 formulae refer to branches. They say the bottom is a source iff the top is a source; the side is a source iff the top is a source and the bottom is not a sink; the top is a sink iff the bottom or side is a sink; and a branch is always open. For any connected points X and Y, X is a source (resp. sink) iff Y is a source (resp. sink). Finally, an inlet (resp. outlet) is a source (resp. sink) which has no incoming (resp. outgoing) connections.

[1] If a point is a source and a sink, then fluid is flowing through there (e.g. $side(b1)$). If it is a source but not a sink, then fluid is stagnating there (e.g. $bottom(b1)$).

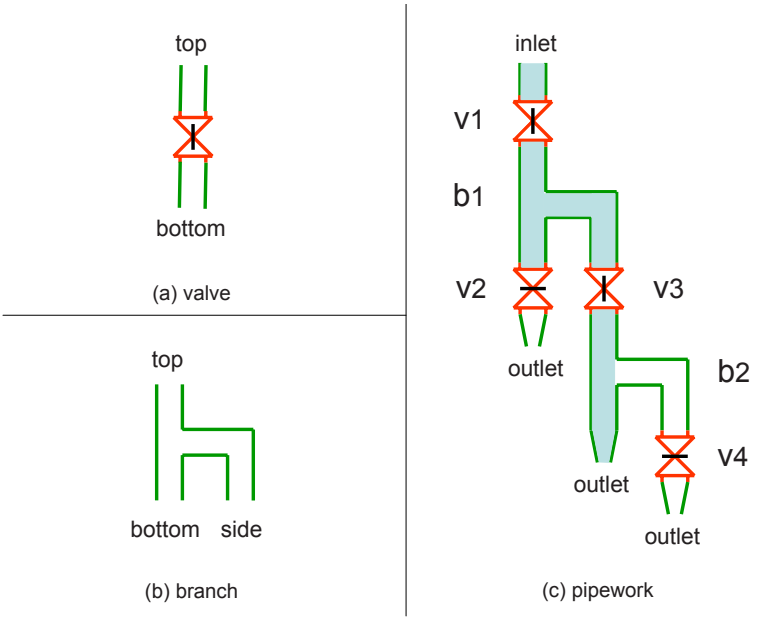


Fig. 1. Valves, branches, and pipework layout for Example 2

The component and connection information for Figure 2 is given by the scenario description S in Figure 3, which is a conjunction of 15 ground atoms. Note how this description does not include the state of each valve as we wish to induce a hypothesis specifying precisely this. In particular we seek a hypothesis specifying which valves are open in order to explain the observation that fluid is flowing from the bottom of branch $b2$. If we suspect the valve state may depend upon whether its bottom is an outlet or not, and we require that $v4$ is not open, then we have the learning problem in Figure 4. Given this task, we would like to learn the hypothesis H stating that a valve is open iff its bottom is not an outlet. It is easy to check $H \in \mathcal{L}_M$ and a classical theorem prover can verify $B \wedge H \models E^+$ and $B \wedge H \wedge E^- \not\models \bot$. Thus H is a correct ILP hypothesis.

Transforming B to clausal form is a trivial operation that results in 21 Horn clauses and 2 non-Horn clauses, all with no more than 4 literals. One of the non-Horn clauses is $sink(bottom(X)) \vee sink(side(X)) \leftarrow sink(top(X)) \wedge branch(X)$. This clause is produced by the 5th formula in D, which models the preferential behaviour of a fluid to flow through the bottom of a branch instead of the side. It says the top of a branch is a sink only if the bottom or the side of the branch is a sink. This knowledge is disjunctive: if the top is a sink, then we know either the bottom is a sink or the side is a sink but, without further information, we may not know which. Given the non-Horn nature of B and H, a full clausal ILP approach is needed to solve this problem. Given the strong bias specified by M, this approach should utilise M in the learning process.

$$
\begin{aligned}
valve(X) &\rightarrow (\, source(bottom(X)) \leftrightarrow open(X) \wedge source(top(X)) \,) \\
valve(X) &\rightarrow (\, sink(top(X)) \leftrightarrow open(X) \wedge sink(bottom(X)) \,) \\
branch(X) &\rightarrow (\, source(bottom(X)) \leftrightarrow source(top(X)) \,) \\
branch(X) &\rightarrow (\, source(side(X)) \leftrightarrow source(top(X)) \wedge \neg sink(bottom(X)) \,) \\
branch(X) &\rightarrow (\, sink(top(X)) \leftrightarrow sink(bottom(X)) \vee sink(side(X)) \,) \\
branch(X) &\rightarrow (\, open(X) \,) \\
connect(X,Y) &\rightarrow (\, source(Y) \leftrightarrow source(X) \,) \\
connect(X,Y) &\rightarrow (\, sink(X) \leftrightarrow sink(Y) \,) \\
inlet(X) &\rightarrow (\, source(X) \,) \\
inlet(X) &\rightarrow (\, \neg connect(Y,X) \,) \\
outlet(X) &\rightarrow (\, sink(X) \,) \\
outlet(X) &\rightarrow (\, \neg connect(X,Y) \,)
\end{aligned}
$$

Fig. 2. Domain knowledge (D) for Example 2

$$
\begin{gathered}
branch(b1) \wedge branch(b2) \\
valve(v1) \wedge valve(v2) \wedge valve(v3) \wedge valve(v4) \\
inlet(top(v1)) \\
outlet(bottom(b2)) \wedge outlet(bottom(v2)) \wedge outlet(bottom(v4)) \\
connect(bottom(v1),\, top(b1)) \\
connect(bottom(b1),\, top(v2)) \\
connect(side(b1),\, top(v3)) \\
connect(bottom(v3),\, top(b2)) \\
connect(side(b2),\, top(v4))
\end{gathered}
$$

Fig. 3. Scenario description (S) for Example 2

$$
\begin{aligned}
B &= D \cup S \\
E^{+} &= \{source(bottom(b2))\} \\
E^{-} &= \{\neg open(v4)\} \\
M &= \left\{ \begin{array}{l} modef(*, open(+)) \\ modef(*, \neg open(+)) \end{array} \right\} \cup \left\{ \begin{array}{l} moder(*, outlet(bottom(+))) \\ moder(*, \neg outlet(bottom(+))) \\ moder(*, \neg valve(+)) \end{array} \right\} \\
H &= \left\{ \begin{array}{l} open(X) \vee outlet(bottom(X)) \vee \neg valve(X). \\ \neg open(X) \vee \neg outlet(bottom(X)) \vee \neg valve(X). \end{array} \right\}
\end{aligned}
$$

Fig. 4. Inputs (B, E^{+}, E^{-}, M) and Output (H) for Example 2

4 Full Clausal Hybrid Abductive Inductive Learning

This section explains how SOLAR can be used to lift the HAIL methodology from the Horn case to full clausal logic. In particular, it describes how SOLAR is used in each phase of the HAIL learning cycle to implement the reasoning tasks of abduction and query answering, as well as coverage and consistency checking. Hereafter, we will write $Solar(T, P)$ to denote the set of consequences returned by SOLAR for a theory T and production field P. To ensure termination, we also assume an implicit bound on the depth of any resolution derivation.

First we explain how SOLAR can be used for abduction in order to compute the head literals of a Kernel Set. Given a background theory B, Skolemised seed example $E\sigma$, and mode declarations M, this involves computing a unit theory $\Delta = \{L_1, \ldots, L_m\}$ such that $B \wedge \Delta \models E\sigma$ and each L_i is an instance of a head declaration scheme in M. This is equivalent to the following abductive task. Given a theory T, a ground clause C (called a goal) and a set of literals Ls (whose instances are called abducibles), find a theory Δ (explanation) such that $T \wedge \Delta \models C$, each literal L_i is subsumed by a literal in Ls. We are interested in the subsume-minimal explanations, as these lead to smaller Kernel Sets that are easier to generalise.

From previous work [2] and [9] it is known that SOLAR can compute such explanations by simply negating the characteristic clauses of the theory $T \wedge \neg C$ and production field $P = \{\overline{L} \,|\, L \in Ls\}$. Furthermore, if Ls is closed under instantiation, then P will be stable and these clauses can be efficiently computed under the pruning strategies described in [5]. But, we wish to use the head declarations to constrain the function symbols appearing in any abduced literals, which will violate the stability requirement of SOL and lead to incompleteness.

Example 3. If $B = \{p \vee \neg a(X, Y) \vee \neg q(X) \vee \neg q(Y)\} \cup \{q(0)\} \cup \{q(1)\}$ and $E = p$ and $M = \{modef(1, a(0, 1))\}$, then the only stable production fields which give the explanation $\Delta = \{a(0, 1)\}$ by the method in [2] must contain $\neg a(X, Y)$. But these also allow the explanations $a(0, 0)$, $a(1, 0)$, and $a(1, 1)$, which violate M. By contrast, the unstable production field $P = \langle\{a(0, 1)\}\rangle$ gives no solutions according to Definition 1, even though Δ is still a correct hypothesis.

To avoid an inefficient generate-and-test approach, we developed a way to transform an unstable production field into an equivalent stable one. In this way, we constrain the abduced atoms without compromising the completeness of SOLAR.The transformation is based on the introduction of a new predicate p_L to represent each literal L in Ls. One clause expressing the relationship between p_L and L is added to T along with the negation of C; and one maximally general instance of p_L is then added to the production field $P = \langle Ms\rangle$. Any bindings computed by SOLAR are propagated back to the literals they represent. The result is a set S of minimal abductive explanations. This is formalised in the *Abduce* procedure below, which is sound and complete for computing subsume-minimal abductive explanations (although it does not necessarily guarantee the consistency of Δ and T, since a stronger consistency check must be performed by fc-HAIL in any case).

Procedure $Abduce(T, C, Ls)$
initialise $B = T \cup \{\overline{G} \,|\, G \in C\}$; $Ms = \emptyset$; $S = \emptyset$
for each literal $L \in Ls$ with variables $X_1, \ldots, X_n$ {
let p_L be a fresh predicate of arity n
let $B = B \cup \{L \vee p_L(X_1, \ldots, X_n)\}$
let $Ms = Ms \cup \{p_L(X_1, \ldots, X_n)\}$
}
let $P = \langle Ms \rangle$
for each clause $\Delta \in Solar(B, P)$ {
let $S = S \cup \{\{L\sigma \,|\, L \vee p_L(X_1, \ldots, X_n) \in B \text{ and } p_L(X_1, \ldots, X_n)\sigma \in \Delta\}\}$
}
return S

Our next task is to use SOLAR to implement the query answering engine used to compute the Kernel Set body literals. Given a theory T and a literal G, we wish to compute the instances of G that are entailed by T. From previous work [4], it is known SOLAR can efficiently compute such instances using the method of Answer Literals [1]. As formalised in the *Query* procedure below, this involves the use of a literal $ans(X_1, \ldots, X_n)$ to represent the variables $X_1, \ldots, X_n$ in G. The production field P is set to return any bindings to these literals (the length restriction avoids the computation of unnecessary disjunctive answers [15]). The bindings D are extracted from the literals computed by SOLAR and propagated back into the goal literal G to obtain the instances A of G entailed by T.

Procedure $Query(T, G)$
let $C = \overline{G} \vee ans(X_1, \ldots, X_n)$ where $X_1, \ldots, X_n$ are the variables in G
let $P = \langle \{ans(X_1, \ldots, X_n)\}, length \leq 1 \rangle$
let $S = Solar(T \cup \{C\}, P)$
let $D = \{\{X_1/t_1, \ldots, X_n/t_n\} \,|\, ans(t_1, \ldots, t_n) \in S\}$
let $A = \{G\sigma \,|\, \sigma \in D\}$
return A

The *Query* procedure is utilised in a full clausal generalisation of the Progol Bottom Set procedure [7], which uses a theory T and a set M of mode declarations to saturate a given head literal L with body literals up to some variable depth bounded by an integer d. As formalised in the *Saturate* procedure below, it maintains a growing set of input terms I that are used to replace + placemarkers in the body declaration schemes. This results in a set of queries whose successful instances result in the insertion of new body literals to the clause X being constructed and the addition of new terms to I. Aside from the fact it uses a full clausal query engine, this is exactly the same procedure used by MDIE systems such as Aleph and Progol.

Procedure $Saturate(T, L, M)$
let $X = \{L\}$
let D be a head declaration in M whose schema subsumes L
let I be the set of terms in L corresponding to $+$ place-markers in D
for each integer i from 1 up to some depth d {
for each body declaration C in M {
let Q be the set of literals obtained from the scheme S of C by replacing all $+$ place-markers by terms from I (in all possible ways) and replacing all other place-markers by distinct variables
let $A = \bigcup_{q \in Q} Query(T, q)$
add to X all literals in A
add to I all terms in A corresponding to $-$ place-markers in C
}}
return X

All these procedures are used in the fc-HAIL learning cycle, formalised below. Like Progol, it uses a covering loop to incrementally construct a hypothesis H using one uncovered example E at a time to focus the generalisation process. Here, we note that coverage and consistency checking can easily be performed by SOLAR since a theory T is inconsistent iff $\emptyset \in Abduce(T, \emptyset, \emptyset)$; and a theory T entails a clause C iff $T \wedge \neg C$ is inconsistent.

If it is non-ground, the seed example E is Skolemised by a substitution σ binding each variable to a fresh constant. A minimal abductive explanation Δ is then computed for the goal $E\sigma$ using the abducibles obtained from the head declaration schemas. Each atom of Δ is saturated with instances of the body declaration schemas to obtain a Kernel Set K. This is generalised to obtain a consistent subsuming hypothesis H.[2]

Procedure $fcHAIL(B, E^+, E^-, M)$
let $H = \emptyset$
let Ls be the set of head declaration schemas
while $B \cup H$ does not cover E^+ {
let E be any seed example $E \in E^+$ not covered by $B \cup H$
let σ be any Skolemising substitution for E
let Δ be any abductive explanation $\Delta \in Abduce(B \cup H, E\sigma, Ls)$ that is consistent with $B \wedge H \wedge E^+ \wedge E^-$
let K be the Kernel Set $K = \bigcup_{L \in \Delta} Saturate(T \cup \{\overline{X}\sigma \mid X \in E\}, L, M)$
add to H any consistent hypothesis H' where $H' \in \mathcal{L}_M$ and $H' \geq K$
}
return H

[2] We currently perform this generalisation by applying the Progol A^* search procedure, in turn, to each clause in K. This ensures that, in the Horn case, we always compute a hypothesis equally or more compressive than any answer returned by Progol.

Example 4. Consider B, E^+, E^- and M in Figure 4. Applying fc-HAIL, we have $Ls = \{\neg open(X), open(X)\}$. We must choose $E = source(bottom(b2))$. As E is ground we have $\sigma = \emptyset$. There is one consistent minimal abductive explanation

$$\Delta = \{open(v1), \neg open(v2), open(v3)\}$$

which results in the unique Kernel Set

$$K = \left\{ \begin{array}{l} open(v1) \vee outlet(bottom(v1)) \vee \neg valve(v1). \\ \neg open(v2) \vee \neg outlet(bottom(v2)) \vee \neg valve(v2). \\ open(v3) \vee outlet(bottom(v3)) \vee \neg valve(v3). \end{array} \right\}$$

As required, this is subsumed by the hypothesis

$$H = \left\{ \begin{array}{l} open(X) \vee outlet(bottom(X)) \vee \neg valve(X). \\ \neg open(X) \vee \neg outlet(bottom(X)) \vee \neg valve(X). \end{array} \right\}$$

Termination of fc-HAIL is ensured by bounding various parameters such as the number of abducibles in Δ and the depth of resolution derivations. This also ensures the worst case time complexity of HAIL is polynomial in n.

5 Related Work

Previous work on inverse entailment can be partitioned into two classes: (A) efficient and incomplete Horn clause systems, which use greedy covering loops and strong forms of language and search bias, such as Progol [7] and Aleph [14]; and (B) inefficient but complete full clausal approaches, which use weaker forms of language and search bias, such as CF-Induction [3] and the Residue Procedure [17]. We see fc-HAIL as applying the efficient Horn techniques in (A) to the more expressive full clausal setting of (B). We now use Example 2 to briefly compare fc-HAIL with these related approaches.

Progol and Aleph are direct Horn implementations of the MDIE methodology described in Section 2.3. Given the inputs B, E^+, E^- and M in Figure 4, they cannot compute H as (i) B and H contain non-Horn clauses, (ii) it is necessary to infer two hypothesis clauses in order to explain a single example clause, and (iii) H does not entail any Bottom Set of B and E.

CF-Induction computes hypotheses by generalising a theory F obtained by negating a set CC of instances of characteristic clauses of $B \wedge \neg E$. In our example, the smallest possible theory F from which CF-Induction can infer the hypothesis H is shown below.

$$F = \left\{ \begin{array}{l} open(v1) \vee \neg outlet(bottom(v2)) \vee outlet(bottom(v3)) \vee \\ \quad outlet(bottom(v1)) \vee \neg valve(v3) \vee \neg valve(v2) \vee \neg valve(v1). \\ \\ \neg open(v2) \vee \neg outlet(bottom(v2)) \vee outlet(bottom(v3)) \vee \\ \quad outlet(bottom(v1)) \vee \neg valve(v3) \vee \neg valve(v2) \vee \neg valve(v1). \\ \\ open(v3) \vee \neg outlet(bottom(v2)) \vee outlet(bottom(v3)) \vee \\ \quad outlet(bottom(v1)) \vee \neg valve(v3) \vee \neg valve(v2) \vee \neg valve(v1). \end{array} \right\}$$

The theory F is much larger than the Kernel Set K and is therefore harder to generalise. Moreover, CF-Induction must work correspondingly harder than HAIL to construct F. In this case, CF-Induction computes a total of 32 characteristic clauses of which 7 must be selected (by the user, from a menu) for CC, while HAIL computes 1 explanation and executes 6 ground queries. This suggests that fc-HAIL is able to make more effective use of language bias than CF-Induction. By contrast, the Residue Procedure, which does not include any form of language bias, must generalise a Residue Hypothesis that is too large to be included in this paper. However, CF-Induction and the Residue Procedure both overcome an inherent incompleteness of MDIE, identified in [16], that is only partially addressed by fc-HAIL. In particular, just like Progol and Aleph, fc-HAIL is unable to solve the Horn clause example presented in [16].

6 Conclusions

This paper proposed a mode directed ILP procedure, called fc-HAIL, for full clausal logic. In particular, it used the SOLAR consequence finding engine to implement the HAIL learning cycle in a way that directly generalises efficient MDIE techniques from Horn clause logic to the more expressive general case. In this way, fc-HAIL exploits language bias more effectively than previous work on non-Horn ILP and removes the need for interactive user assistance. This was achieved, in part, by showing how to overcome the restriction to stable production fields that is assumed in all previous work on SOL. For this purpose, we introduced an efficient transformation that results in only one extra resolution step per abduced atom. A generalisation of this transformation has since been developed to allow full clausal abduction with non-ground abducibles, disjunctive answer extraction, and consistency checking [13].

The potential utility of fc-HAIL was illustrated with a case study modelling the flow of a fluid through a system of pipework. This example suggests that fc-HAIL's ability to automatically compute multi-clause non-Horn hypotheses in response to a single example may be beneficial in practice. We believe that non-Horn ILP will be useful in real applications to enable the representation and discovery of indefinite and disjunctive knowledge. We intend to test this claim using fc-HAIL by extending our model of fluid flow into a model of metabolic flux in biochemical networks. We also aim to better understand the trade-offs between expressivity and efficiency by investigating stronger forms of bias and more powerful inference methods capable of overcoming the incompleteness of fc-HAIL and other methods noted in [16].

Acknowledgements

This work is supported by Research Councils UK (RCUK) and the Japan Society for the Promotion of Science (JSPS). The authors thank the reviewers for their useful comments.

References

1. Green, C.: Theorem-proving by resolution as a basis for question-answering systems. Machine Intelligence 4, 183–205 (1969)
2. Inoue, K.: Linear resolution for Consequence Finding. Artificial Intelligence 56(2-3), 301–353 (1992)
3. Inoue, K.: Induction as Consequence Finding. Machine Learning 55(2), 109–135 (2004)
4. 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)
5. Iwanuma, K., Inoue, K., Satoh, K.: Completeness of Pruning Methods for Consequence Finding Procedure SOL. In: Proceedings of the 3rd International Workshop on First-Order Theorem Proving, pp. 89–100 (2000)
6. Loveland, D.W.: Automated Theorem Proving: A Logical Basis. North-Holland, Amsterdam (1978)
7. Muggleton, S.H.: Inverse Entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming 13(3-4), 245–286 (1995)
8. 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)
9. 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)
10. Ray, O.: Hybrid Abductive-Inductive Learning. PhD thesis, Dept. of Computing, Imperial College (2005), http://www.bcs.org/upload/pdf/oray.pdf
11. Ray, O., Broda, K., Russo, A.M.: Hybrid Abductive Inductive Learning: a Generalisation of Progol. In: Horváth, T., Yamamoto, A. (eds.) ILP 2003. LNCS (LNAI), vol. 2835, pp. 311–328. Springer, Heidelberg (2003)
12. Ray, O., Broda, K., Russo, A.M.: Generalised Kernel Sets for Inverse Entailment. In: Demoen, B., Lifschitz, V. (eds.) ICLP 2004. LNCS, vol. 3132, pp. 165–179. Springer, Heidelberg (2004)
13. Ray, O., Inoue, K.: A consequence finding approach for full clausal abduction. In: Proceedings of the 10th International Conference on Discovery Science (to appear, 2007)
14. Srinivasan, A.: The Aleph Manual (version 4) (2003), http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/index.html
15. Stickel, M.E.: A Prolog technology theorem prover: A New Exposition and Implementation in Prolog. Theoretical Computer Science 104(1), 109–128 (1992)
16. Yamamoto, A.: Which hypotheses can be found with Inverse Entailment? In: Džeroski, S., Lavrač, N. (eds.) ILP 1997. LNCS, vol. 1297, pp. 296–308. Springer, Heidelberg (1997)
17. Yamamoto, A.: Hypothesis finding based on upward refinement of residue hypotheses. Theoretical Computer Science 298, 5–19 (2003)

Mining of Frequent Block Preserving Outerplanar Graph Structured Patterns

Yosuke Sasaki[1], Hitoshi Yamasaki[1], Takayoshi Shoudai[1], and Tomoyuki Uchida[2]

[1] Department of Informatics, Kyushu University
Fukuoka 819-0395, Japan
{yosuke.sasaki,h-yama,shoudai}@i.kyushu-u.ac.jp
[2] Department of Intelligent Systems, Hiroshima City University
Hiroshima 731-3194, Japan
uchida@cs.hiroshima-cu.ac.jp

Abstract. An outerplanar graph is a planar graph which can be embedded in the plane in such a way that all of vertices lie on the outer boundary. Many semi-structured data like the NCI dataset having about 250,000 chemical compounds can be expressed by outerplanar graphs. In this paper, we consider a data mining problem of extracting structural features from semi-structured data. First of all, we define a block preserving outerplanar graph pattern as an outerplanar graph having structured variables. Then, we present an effective Apriori-like algorithm for enumerating frequent block preserving outerplanar graph patterns from semi-structured data in incremental polynomial time. Lastly, by reporting some preliminary experimental results on a subset of the NCI dataset, we evaluate the performance of our algorithms.

Keywords: pattern discovery, graph mining, graph structured pattern, outerplanar graph.

1 Introduction

Large amount of data having graph structures, called semi-structured data, such as map data, CAD, biomolecular, chemical molecules, the World Wide Web are stored in databases. In the fields of Web mining and graph mining, many Web documents and many chemical compounds can be represented by ordered trees and outerplanar graphs, respectively. For example, 94.3% of all elements in the NCI dataset [4], which is one of popular graph mining datasets, are expressed by outerplanar graphs. Outerplanar graphs are planar graphs which can be embedded in the plane in such a way that all of vertices lie on the outer boundary. In Fig. 1, we give four outerplanar graphs G, g_1, g_2, g_3 as examples of outerplanar graphs. In analyzing semi-structured data, we must often solve a subgraph isomorphism problem, which is known to be NP-complete in general. However, subgraph isomorphism problems on some classes of graphs, including trees and biconnected outerplanar graphs, can be solved in polynomial time [2,5]. Moreover, graph isomorphism problems tend to be easier than subgraph isomorphism

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 239–253, 2008.

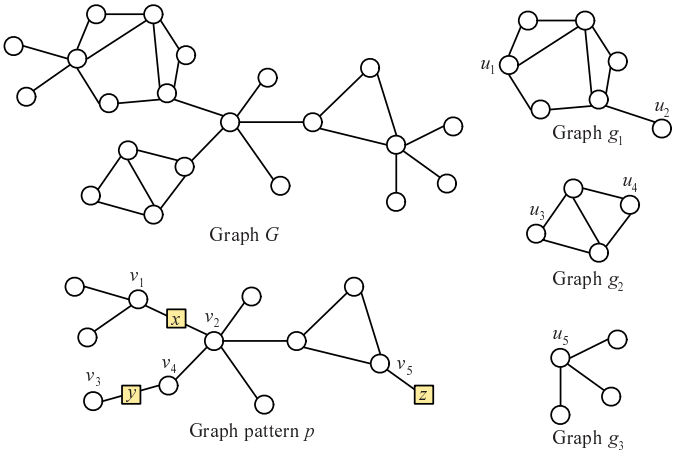


Fig. 1. Outerplanar graphs G, g_1, g_2, g_3 and a block preserving outerplanar graph pattern p. A variable is drawn by a box with lines to its elements. The label inside a box represents a variable label.

problems and are computed in polynomial time for the classes of interval graphs, circular-arc graphs, and planar graphs.

Based on these facts, in this paper, we consider a graph mining problem of extracting structural features from semi-structured data having outerplanar graph structures. In order to solve this graph mining problem, first of all, we define a new graph pattern, called a *block preserving outerplanar graph pattern* (*bpo-graph pattern* for short) as a connected outerplanar graph having structured variables. A variable is a list of at most 2 vertices and can be replaced with an arbitrary connected outerplanar graph.

We say that a bpo-graph pattern p *matches* an outerplanar graph G if G is obtained from p by replacing all variables with arbitrary connected outerplanar graphs. In Fig. 1, as an example of bpo-graph patterns, we give a bpo-graph pattern p having variables (v_1, v_2), (v_3, v_4), (v_5) labeled with variable labels x, y, z, respectively. The bpo-graph pattern p matches the outerplanar graph G in Fig. 1 obtained by replacing variables (v_1, v_2), (v_3, v_4), (v_5) with outerplanar graphs g_1, g_2, g_3, respectively.

Our goal of this paper is to present an effective algorithm of enumerating all frequent bpo-graph patterns from a given finite set D of outerplanar graphs. It is natural that for a given finite set D of outerplanar graphs, bpo-graph patterns, which are frequent with respect to the number of outerplanar graphs in D which are matched by p, are characteristic in D. As our approaches, in a similar way to a *block and bridge tree* given in [1], we introduce a *block tree* $t(G)$ and a *block tree pattern* $t(p)$ according to an outerplanar graph G and a bpo-graph pattern p, respectively. Then we reduce a problem of deciding whether or not a

bpo-graph pattern p matches an outerplanar graph G to a problem of deciding whether or not the tree pattern $t(p)$ corresponding to p matches the tree $t(G)$ corresponding to G. By giving an Apriori-like algorithm of generating candidate block tree patterns, we enumerate all frequent bpo-graph patterns in incremental polynomial time.

Horváth et al. [1] proposed an Apriori-like algorithm, which works in incremental polynomial time, for enumerating frequent subgraphs appearing in a restricted class of outerplanar graphs, called d-tenuous outerplanar graphs. Then by applying their algorithm to the NCI dataset [4], they found typical subgraph structures of chemical compounds. Yamasaki and Shoudai [10] proposed an interval graph pattern and presented a polynomial time algorithm for finding a minimally generalized interval graph pattern explaining a given finite set of interval graphs. As other related works, in the framework of inductive inference, by Suzuki et al. [7] and Takami et al. [8] gave polynomial time learning algorithms for tree patterns with internal structured variables and two-terminal series parallel graph patterns, respectively.

This paper is organized as follows. In Section 2, we introduce a graph pattern based on [9] and, in Section 3, a bpo-graph pattern as an outerplanar graph having structured variables. In Section 4, we propose a polynomial time algorithm which solves a matching problem between bpo-graph patterns and outerplanar graphs. In Section 5, we propose an Apriori-like algorithm which enumerates all frequent bpo-graph patterns from a finite set of outerplanar graphs. Lastly, by reporting an experimental result of applying our algorithms to a subset of the NCI dataset, we evaluate the performance of our algorithm.

2 Graph Pattern

For a set and a list S, the number of elements in S is denoted by $|S|$. Let Λ and Δ be two alphabets each of whose elements is called a *vertex label* and an *edge label*, respectively. Let X be an infinite alphabet with $X \cap (\Lambda \cup \Delta) = \emptyset$. An element in X is called a *variable label*. In this paper, an undirected graph G is called a (Λ, Δ)-*labeled graph* if all vertices and edges in G are labeled with symbols in Λ and Δ, respectively. We denote by $V(G)$ the set of vertices in G and by $E(G)$ the set of edges in G. In the same way as a term graph presented in [9] which can represent characteristic common structures in graph-structured data, we define a *graph pattern* as follows.

Definition 1 (Graph pattern). A triplet $p = (V, E, H)$ is called a (Λ, Δ)-*graph pattern* if (V, E) is a (Λ, Δ)-labeled graph and H is a set of lists of distinct vertices in V. For a (Λ, Δ)-graph pattern p, we denote by $V(p)$, $E(p)$ and $H(p)$ the sets of all vertices, edges and variables of p, respectively. For a vertex v of p, we assign a vertex label $\lambda_p(v) \in \Lambda$ to v, that is, λ_p is a vertex labeling from $V(p)$ to Λ. For an edge e of p, we also assign an edge label $\delta_p(e) \in \Delta$ to e, that is, δ_p is an edge labeling from $E(p)$ to Δ. An element in H is called a *variable* of p and each vertex in a variable is called a *port* of the variable. For example, for a variable $h = (v_1, v_2, \ldots, v_\ell) \in H$, $v_1, v_2, \ldots, v_\ell$ are pots of h. If a variable

has only one port, it is called a *terminal variable*, otherwise called an *internal variable*. All variables in H are labeled with variable labels in X.

Below (Λ, Δ)-labeled graphs and (Λ, Δ)-graph patterns are simply called labeled graphs and graph patterns, respectively, when the label sets Λ and Δ are clear from the context. The *degree* of v, denoted by $d_p(v)$, is the total sum of edges adjacent to v and internal variables including v, that is $d_p(v) = |\{\{u, v\} \mid \{u, v\} \in E(p)\} \cup \{h \mid h \in H(p),\ |h| = 2,\ h \text{ contains } v\}|$.

A *graph pattern* p is called a *linear (or regular) graph pattern* if all variables in $H(p)$ have mutually distinct variable labels in X. Let p and q be linear graph patterns. We say that p is *isomorphic* to q if there exists a bijection $\psi : V(p) \to V(q)$ such that (1) for any $v \in V(p)$, $\lambda_p(v) = \lambda_q(\psi(v))$, (2) $\{u, v\} \in E(p)$ if and only if $\{\psi(u), \psi(v)\} \in E(q)$, (3) for any $\{u, v\} \in E(p)$, $\delta_p(\{u, v\}) = \delta_q(\{\psi(u), \psi(v)\})$, and (4) $(v_1, \ldots, v_\ell) \in H(p)$ if and only if $(\psi(v_1), \ldots, \psi(v_\ell)) \in H(q)$, where $\ell \geq 1$.

We assume that all graph patterns in this paper are linear. Then we call linear graph patterns graph patterns simply.

Definition 2 (Binding and Substitutions). Let p and q be graph patterns and x a variable label in X. Let $\sigma = (u_1, \ldots, u_k)$ be a list of k distinct vertices in q. The form $x := [q, \sigma]$ is called a *binding* for x. We can apply a binding $x := [q, \sigma]$ to a variable $h = (v_1, \ldots, v_\ell)$ in p which is labeled with x if the binding $x := [q, \sigma]$ satisfies that (1) $\ell = k$ and (2) $\lambda_q(u_i) = \lambda_p(v_i)$ for all i $(1 \leq i \leq \ell = k)$. A new graph pattern $p\{x := [q, \sigma]\}$ is obtained by applying the binding $x := [q, \sigma]$ to p in the following way. Let $h = (v_1, \ldots, v_\ell)$ be a variable in p with the variable label x. Let q' be one copy of q and $u'_1, \ldots, u'_k$ the vertices of q' corresponding to $u_1, \ldots, u_k$ of q, respectively. For the variable $h = (v_1, \ldots, v_\ell)$, we attach q' to p by removing the variable h from $H(p)$ and by identifying the vertices $v_1, \ldots, v_\ell$ with the vertices $u'_1, \ldots, u'_k$ of q', respectively.

A *substitution* is a finite collection of bindings $\{x_1 := [q_1, \sigma_1], \cdots, x_m := [q_m, \sigma_m]\}$, where $x_1, \ldots, x_m$ are mutually distinct variable labels in X. For a graph pattern p and a substitution θ, $p\theta$ denotes the graph pattern obtained from p and θ by applying all the bindings in θ to p simultaneously.

Below we regard all labeled graphs as graph patterns without variables. As an example of graph patterns, in Fig. 1, we give a graph pattern p having variables (v_1, v_2), (v_3, v_4), (v_5) labeled with x, y, z, respectively, so that the graph pattern $p\{x := [g_1, (u_1, u_2)], y := [g_2, (u_3, u_4)], z := [g_3, (u_5)]\}$ is isomorphic to the graph G in Fig. 1 where g_1, g_2, g_3 are labeled graphs in Fig. 1.

3 Block Preserving Outerplanar Graph Patterns and Block Tree Patterns

Let G be a connected labeled graph. For a subset U of $V(G)$, the *induced subgraph* of G by U, denoted by $G[U]$, is a subgraph $G[U] = (U, \{\{u, v\} \in E(G) \mid$ both u and v are in $U\})$ of G. For a vertex v in $V(G)$, v is called a *cutpoint* of

G if $G[V(G) - \{v\}]$ is unconnected. G is said to be *biconnected* if G has no cutpoint. A maximal connected subgraph without a cutpoint is called a *block* (or a *biconnected component*). An edge which does not belong to any block is called a *bridge*. An *outerplanar labeled graph* is a planar labeled graph which can be embedded in the plane so that all vertices lie on the border of the same face. We denote by $\mathcal{O}$ the set of all outerplanar labeled graphs. Since any block B of an outerplanar labeled graph has a unique cycle which contains all vertices of B, we call the cycle of B the *Hamiltonian cycle* of B. A *diagonal* is an edge in B which is not in the Hamiltonian cycle of B. Consider an example of blocks given in Fig. 2. The block B has four diagonals $\{u_2, u_4\}$, $\{u_2, u_7\}$, $\{u_4, u_7\}$, $\{u_5, u_7\}$.

In order to make our discussion simpler, we assume that all outerplanar labeled graphs are connected. We can easily extend our results of this paper to the case without the assumption.

Definition 3 (Block preserving outerplanar graph patterns). A graph pattern p is a *block preserving outerplanar graph pattern* (*bpo-graph pattern* for short) if p satisfies the following three conditions.

1. Any internal variable has just 2 ports.
2. $(V(p), E(p) \cup \{\{u, v\} \mid (u, v) \in H(p)\})$ is an outerplanar labeled graph.
3. Each port of any internal variable in p is either a cutpoint or a vertex whose degree is just one.

Since all internal variables in a bpo-graph pattern are bridges in G_p, we call an internal variable in a bpo-graph pattern a *bridge variable*. For example, a graph pattern p in Fig. 1 is a bpo-graph pattern and two variables (v_1, v_2) and (v_3, v_4) of p are bridge variables. We denote by $\mathcal{OP}$ the set of all bpo-graph patterns. For a bpo-graph pattern $p \in \mathcal{OP}$ and an outerplanar labeled graph $G \in \mathcal{O}$, we say that p *matches* G if there exists a substitution θ such that $p\theta$ is isomorphic to G.

Horváth et al. [1] proposed a special data structure, called a block-bridge tree, for representing connections among blocks of an outerplanar labeled graph. In a similar way to a block-bridge tree, for a bpo-graph pattern p, we define a new tree-structured pattern, called a *block tree pattern* of p as follows.

Definition 4 (Block tree patterns and block trees). Let p be a bpo-graph pattern in $\mathcal{OP}$ and $\mathcal{B}(p)$ the set of all blocks of p. For each block B of p, we assign a new vertex not in $V(p)$ to B which is denoted by v_B and is called a *block vertex* of B. A *block tree pattern of* p, denoted by $t(p)$, is a graph pattern (V_p, E_p, H_p), where

$$V_p = V(p) \cup \{v_B \mid B \in \mathcal{B}(p)\},$$

$$E_p = \left(E(p) - \bigcup_{B \in \mathcal{B}(p)} E(B) \right) \cup \bigcup_{B \in \mathcal{B}(p)} \{\{v_B, v\} \mid v \in V(B)\}, \text{ and}$$

$$H_p = H(p).$$

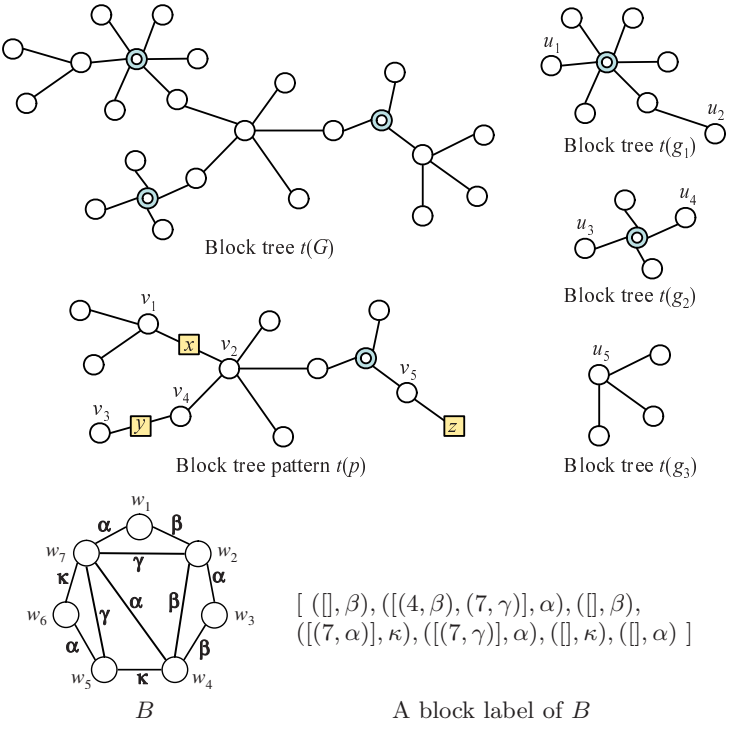


Fig. 2. Block trees $t(G)$, $t(g_1)$, $t(g_2)$, $t(g_3)$, the block tree pattern $t(p)$, a block B and its block label. A block vertex is drawn by a double circle.

Let "#" be a symbol not in $\Lambda \cup \Delta$. We define vertex labels of $t(p)$ as follows: $\lambda_{t(p)}(v) = \lambda_p(v)$ if $v \in V(p)$, $\lambda_{t(p)}(v) = \#$ otherwise. In a similar way, we define edge labels of $t(p)$ as follows: $\delta_{t(p)}(e) = \delta_p(e)$ if $e \in E(p)$, $\delta_{t(p)}(e) = \#$ otherwise. We call a block tree pattern without variables a *block tree*, simply.

For example, for the bpo-graph pattern p and outerplanar labeled graphs G, g_1, g_2, g_3 given in Fig. 1, we give the block tree pattern $t(p)$ of p and block trees $t(G)$, $t(g_1)$, $t(g_2)$, $t(g_3)$ of G, g_1, g_2, g_3 in Fig. 2, respectively.

We note that any block in a bpo-graph pattern has the unique Hamiltonian cycle. Hence, for a block B in a bpo-graph pattern p, we can index all vertices of B in the clockwise or anticlockwise order of the Hamiltonian cycle of B. That is, by specifying a vertex in $V(B)$, called a *start vertex*, and a rotation direction of the Hamiltonian cycle of B, e can easily construct a numbering function ρ_B from $V(B)$ to the set $\{1, 2, \ldots, |V(B)|\}$. Hereafter, for a vertex v of a block B, we also use the number $\rho_B(v)$ instead of v as a vertex identifier. For example, for a block B given in Fig. 2, $\rho_B(w_i) = i$ for each i $(1 \leq i \leq 7)$ if all vertices are numbered in the clockwise order of the Hamiltonian cycle of B from the vertex w_1. In the case of the anticlockwise order, $\rho_B(w_1) = 1$ and $\rho_B(w_i) = 9 - i$ for each i $(2 \leq i \leq 7)$.

Let B be a block having ℓ vertices. For any i $(1 \leq i \leq \ell)$, we suppose that the vertex i is adjacent to k_i diagonals $\{i, j_1\}, \{i, j_2\}, \ldots, \{i, j_{k_i}\}$ in B, where $0 \leq k_i \leq \ell - 3$ and $i \leq j_1 < j_2 < \cdots < j_{k_i} \leq \ell$. Then, for a block vertex v_B of B, we define a *block label* of v_B, denoted by $\mu_p(v_B)$, as a list $[(\phi_p(1), \delta_p(\{1,2\})), (\phi_p(2), \delta_p(\{2,3\})), \ldots, (\phi_p(\ell), \delta_p(\{\ell, 1\}))]$, where for each i $(1 \leq i \leq \ell)$, $\phi_p(i) = [(j_1, \delta_p(\{i, j_1\})), (j_2, \delta_p(\{i, j_2\})), \ldots, (j_{k_i}, \delta_p(\{i, j_{k_i}\}))]$. For example, let p be a bpo-graph pattern having a block B given in Fig. 2 as a subgraph. For a block vertex v_B of B, the block label $\mu_p(v_B)$ of v_B is the list $[([], \beta),\ ([(4,\beta),(7,\gamma)], \alpha),\ ([], \beta),\ ([(7,\alpha)], \kappa),\ ([(7,\gamma)], \alpha),\ ([], \kappa),\ ([], \alpha)]$.

4 Matching Algorithm for Block Preserving Outerplanar Graph Patterns

Let p and p' be bpo-graph patterns in $\mathcal{OP}$, and r and r' vertices of p and p', respectively. Let $t(p, r)$ be an unordered tree obtained from $t(p)$ by specifying r as a root. For each block B in p, the block labels of $t(p, r)$ are constructed from $\mu_p(v_B)$ by regarding the nearest vertex from r in B as a start vertex of the Hamiltonian cycle of B. The new block label of a block vertex v_B in $t(p, r)$ is denoted by $\mu_p(v_B, r)$. We denote by $\bar{\mu}_p(v_B, r)$ the block label obtained from $\mu_p(v_B, r)$ by changing the rotation direction of the Hamiltonian cycle. We say that $t(p, r)$ is *equivalent* to $t(p', r')$ if there exists an isomorphism ψ from $t(p, r)$ to $t(p', r')$ such that $r' = \psi(r)$ and for all block vertices v_B in $t(p, r)$, either $\mu_p(v_B, r) = \mu_{p'}(\psi(v_B), r')$ or $\mu_p(v_B, r) = \bar{\mu}_{p'}(\psi(v_B), r')$ holds. We say that $t(p)$ is *equivalent* to $t(p')$ if there exists two vertices r in p and r' in p' such that $t(p, r)$ is equivalent to $t(p', r')$. Let r be a vertex in p and v a vertex of $t(p, r)$. A block tree pattern rooted at v of $t(p, r)$, denoted by $t(p, r)[v]$, is a block tree pattern induced by v and all descendants of v.

We call a block tree pattern with no variable a *block tree*. For a block tree pattern t and a block tree T, we say that t *matches* T if there exists a substitution θ such that all graph patterns appearing in θ are block trees and $t\theta$ is equivalent to T. We show the next lemmas for bpo-graph patterns and block tree patterns. We omit the proofs.

Lemma 1. *Let p and p' be bpo-graph patterns in $\mathcal{OP}$ and r a vertex in p. Let $x := [p', \sigma]$ be a binding for p. Then there exists a vertex r' in σ such that $t(p, r)\{x := [t(p', r'), \sigma]\}$ is equivalent to $t(p\{x := [p', \sigma]\}, r)$.*

Lemma 2. *Let p and q be bpo-graph patterns in $\mathcal{OP}$. Then, p is isomorphic to q if and only if $t(p)$ is equivalent to $t(q)$.*

In this section, we give a polynomial time algorithm for computing the following problem.

Matching Problem for $\mathcal{OP}$
Input: An outerplanar graph $G \in \mathcal{O}$ and a bpo-graph pattern $p \in \mathcal{OP}$.
Problem: Decide whether or not p matches G.

From Lemmas 1 and 2, Matching Problem for $\mathcal{OP}$ can be reduced to deciding a block tree pattern $t(p)$ matches a block tree $t(G)$. Hence, we can present a polynomial time algorithm for solving Matching Problem for $\mathcal{OP}$ by modifying the matching algorithm for tree patterns given in [6].

Let n be the number of vertices in p. Let r be a vertex in G and r' a vertex in p. For all vertices u in $t(G,r)$, we compute a subset of $V(t(p,r'))$, which is called a *correspondence-set* (C-set for short) of u and denoted by $CS(u)$, in the following way. We assume that each C-set is stored by a simple array of length $O(n)$. We compute C-sets of all vertices in $t(G,r)$ in postorder depending on a kind of each vertex. For a vertex u in $t(G,r)$, let $CS_{\mathcal{P}}(u) = \{c' \in CS(c) \mid c \text{ is a child of } u \text{ and } c' \text{ is a port of a variable in } p\}$.

Leaf. For all leaves u of $t(G,r)$, $CS(u)$ is the set of all vertices u' of $t(p,r')$ such that $d_{t(p,r')}(u') = 1$ and $\lambda_p(u') = \lambda_G(u)$.

Block vertex. Let u be a block vertex of $t(G,r)$ corresponding to a block B of G. Let $c_1, \ldots, c_\ell$ be the children of u which appear on the Hamiltonian cycle of B in this order. Let $CS_{\mathcal{B}}(u)$ be the set of all block vertices u' in $t(p,r)$ satisfying the following conditions: u' has just ℓ children $c'_1, \ldots, c'_\ell$ such that they appear on the Hamiltonian cycle of a block represented by u' in this order and either $(\mu_G(u,r) = \mu_p(u',r')$ and $c'_i \in CS(c_i))$ or $(\mu_G(u,r) = \bar{\mu}_p(u',r')$ and $c'_i \in CS(c_{\ell-i+1}))$ holds for $1 \leq i \leq \ell$. This work consumes $O(n\ell)$ time. Finally $CS(u) = CS_{\mathcal{B}}(u) \cup CS_{\mathcal{P}}(u)$.

Otherwise. Let u be a non-block vertex which is not a leaf in $t(G,r)$ and $c_1, \ldots, c_\ell$ the children of u. Let $CS_{\mathcal{NB}}(u)$ be the set of all non-block vertices u' in $t(p,r)$ satisfying the following condition: $\lambda_G(u) = \lambda_p(u')$, u' has at most ℓ children $c'_1, \ldots, c'_{\ell'}$ $(\ell' \leq \ell)$ and there are ℓ' vertices $c_{k_1}, \ldots, c_{k_{\ell'}}$ among $c_1, \ldots, c_\ell$ such that $c'_i \in CS(c_{k_i})$ and $\delta_p(\{u', c'_i\}) = \delta_G(\{u, c_{k_i}\})$ for all $i = 1, \ldots, \ell'$. We can decide whether or not this condition are satisfied for u and u' in the following way. First we construct a bipartite graph (U, V, E) as follows: $U = \{CS(c_1), \ldots, CS(c_\ell)\}$, $V = \{c'_1, \ldots, c'_{\ell'}\}$ and $E = \{(CS(c_i), c'_j) \mid c'_j \in CS(c_i)$ and $\delta_p(\{u', c'_j\}) = \delta_G(\{u, c_i\})$ $(1 \leq i \leq \ell,\ 1 \leq j \leq \ell')\}$. Next we compute a maximum bipartite graph matching problem for (U, V, E). If u' is a port of a variable and the bipartite graph has a matching of size ℓ', or u' is not a port of any variable and the bipartite graph has a matching of size exactly ℓ, we conclude that u and u' satisfy the above condition. We need $O(\ell\ell'\sqrt{\ell + \ell'})$ time to find a maximum matching for the bipartite graph (U, V, E) by Dinic's algorithm. Then we need $O(n\ell\sqrt{\ell})$ time for all non-block vertices in $t(p,r)$. Finally $CS(u) = CS_{\mathcal{NB}}(u) \cup CS_{\mathcal{P}}(u)$.

The correctness of the above algorithm is shown from the following lemmas.

Lemma 3. *Let G and p be an outerplanar graph in $\mathcal{O}$ and a bpo-graph pattern in $\mathcal{OP}$, respectively. Moreover let r and r' be vertices of G and p, respectively, and u and u' vertices in $t(G,r)$ and $t(p,r')$, respectively. Then,*

(1) *$u' \in CS_{\mathcal{B}}(u)$ or $u' \in CS_{\mathcal{NB}}(u) - CS_{\mathcal{P}}(u)$ if and only if $t(p,r')[u']$ matches $t(G,r)[u]$, and*

(2) $u' \in CS_{\mathcal{P}}(u)$ *if and only if there is a descendant* d *of* u *such that* $t(p, r')[u']$ *matches* $t(G, r)[d]$.

Lemma 4. *Let* G *and* p *be an outerplanar graph in* $\mathcal{O}$ *and a bpo-graph pattern in* $\mathcal{OP}$*, respectively. And let* r' *be a vertex in* p*. Then there exists a vertex* r *in* G *such that* $r' \in CS(r)$ *if and only if* p *matches* G*.*

For each vertex $r \in V(G)$, we need $O(nN\sqrt{d})$ time for computing all C-sets for vertices in $t(G, r)$, where N and n are the numbers of vertices in G and p, respectively, and d is the maximum degree of cutpoints in p. Then we have the following theorem.

Theorem 1. *Matching Problem for* $\mathcal{OP}$ *is computable in polynomial time.*

5 Pattern Enumeration Algorithm for Frequent BPO Graph Pattern Problem

Let p and q be bpo-graph patterns in $\mathcal{OP}$ and σ a list of vertices of length one or two in q. We easily show that for a variable h in p labeled with x, a graph pattern $p\{x := [q, \sigma]\}$ is also a bpo-graph pattern in $\mathcal{OP}$. For $p \in \mathcal{OP}$, let $L(p) = \{p\theta \in \mathcal{O} \mid \text{all graph patterns appearing in } \theta \text{ are in } \mathcal{O}\}$. It is easy to see that if Λ and Δ have infinitely many symbols, for p and $p' \in \mathcal{OP}$, $L(p) \subseteq L(p')$ if and only if there is a substitution θ such that all graph patterns appearing in θ are in $\mathcal{OP}$ and p is isomorphic to $p'\theta$.

Let D be a finite subset of $\mathcal{O}$ and p a bpo-graph pattern in $\mathcal{OP}$. Then, we denote by $match_D(p)$ the number of outerplanar labeled graphs in D which are matched by p. The *frequency* of p with respect to D, denoted by $supp_D(p)$, is defined as $supp_D(p) = match_D(p)/|D|$. Let t be a real number where $0 < t \leq 1$. A bpo-graph pattern $p \in \mathcal{OP}$ is *t-frequent* with respect to D if $supp_D(p) \geq t$. We call this real number t a *frequency threshold*.

In this section, we give an Apriori-like algorithm for computing the next problem.

Frequent Block Preserving Outerplanar Graph Pattern Problem
Input: A set of outerplanar labeled graphs $D \subset \mathcal{O}$ and a frequency t $(0 < t \leq 1)$.
Output: The set of all t-frequent bpo-graph patterns in $\mathcal{OP}$ with respect to D.

For an integer $k \geq 0$, a *k-block tree pattern* is defined to be a block tree pattern such that the total sum of the numbers of block vertices, bridge variables, and edges not adjacent to any block vertex is equal to k. Let D be a set of outerplanar labeled graphs in $\mathcal{O}$ and t a real number where $0 < t \leq 1$. Let L_k^t be the set of all t-frequent k-block tree patterns with respect to D and C_k^t a set of candidate k-block tree patterns, which contains L_k^t. Let Λ_D (resp. Δ_D) be the set of all vertex (resp. edge) labels appearing in D. We compute C_k^t and L_k^t $(k \geq 0)$ in the following way.

0-block tree patterns. For all $a \in \Lambda_D$, we make a new vertex v labeled with a to construct a new block tree pattern $p_a = (\{v\}, \emptyset, \{(v)\})$. C_0^t is the set of all block tree patterns p_a obtained from all $a \in \Lambda_D$ in such a way. Let L_0^t be the set of all block tree patterns in C_0^t which are t-frequent.

1-block tree patterns. C_1^t is the set of all block tree patterns obtained from L_0^t in the following three ways 1-(a), 1-(b) and 2. Initially let $C_1^t = \emptyset$.

1. For two 0-block tree patterns $p = (\{v\}, \emptyset, \{(v)\})$ and $p' = (\{v'\}, \emptyset, \{(v')\})$ in L_0^t, two copies $q = (\{w\}, \emptyset, \{(w)\})$ of p and $q' = (\{w'\}, \emptyset, \{(w')\})$ of p' are made. Then,
   (a) for all $s \in \Delta_D$, a new block tree pattern $q_s = (\{w, w'\}, \{\{w, w'\}\}, \{(w), (w')\})$ with an edge $\{w, w'\}$ labeled with s is added to C_1^t, and
   (b) a new block tree pattern $q_x = (\{w, w'\}, \emptyset, \{(w), (w'), (w, w')\})$ is added to C_1^t.
2. For every block B appearing in all outerplanar labeled graphs in D, a new block tree pattern $q_B = (V(t(B)), E(t(B)), \{(w) \mid w$ is a non-block vertex in $V(t(B))\})$ is added to C_1^t.

Let L_1^t be the set of all block tree patterns in C_1^t which are t-frequent.

Let p be a block tree pattern. We say that p' is a *block tree subpattern* of p if p' is a block tree pattern and $V(p') \subseteq V(p)$, $E(p') \subseteq E(p)$, and $H(p') \subseteq H(p)$. Moreover we say that p' is a *terminal block tree subpattern* if p' is a block tree subpattern of p and either of the following forms:

(1) $p' = (\{u, v\}, \{\{u, v\}\}, \{(v)\})$ or $p' = (\{u, v\}, \emptyset, \{(u, v), (v)\})$, where v is a non-block vertex adjacent to only u in p.

(2) $p' = (\{u, v_B, v_1, \ldots, v_\ell\}, \{\{u, v_B\}, \{v_1, v_B\}, \ldots, \{v_\ell, v_B\}\}, \{(v_1), \ldots, (v_\ell)\})$ for some $\ell \geq 2$, where v_B is a block vertex adjacent to only $u, v_1, \ldots, v_\ell$ in p, while all $v_1, \ldots, v_\ell$ are adjacent only to v_B in p.

The vertex u appearing in (1) and (2) is called a *connected point* of p'. For a block tree pattern p and a terminal block tree subpattern p', we denote by $p \ominus p'$ the block tree subpattern obtained from p by removing all vertices in p' except for the connected point of p' and all edges and variables in p'.

k-block tree patterns ($k \geq 2$). Initially let $C_k^t = \emptyset$. For two $(k-1)$-block tree patterns p and q in L_{k-1}^t, let p' and q' be two terminal block tree subpatterns of p and q, respectively. If $p \ominus p'$ is equivalent to $q \ominus q'$ then a new block tree pattern r is constructed in such a way that a copy of p' and a copy of q' are connected to a copy of $p \ominus p'$ through the connected points of p' and q', respectively. The block tree pattern r is added to C_k^t. Let L_k^t be the set of all block tree patterns in C_k^t which are t-frequent.

We formally describe this algorithm in Fig. 3, and in Fig. 4, we give examples of 0-block tree patterns, 1-block tree patterns and k-block tree patterns constructed in the above way. For any $k \geq 0$, the *size* of L_k^t is defined as the total sum of the numbers of vertices of block tree patterns in L_k^t.

Algorithm: FBTPGen;
Input: a set of block trees D and a frequency threshold t $(0 < t \leq 1)$;
Output: the set of all t-frequent block tree patterns of D;
begin
1: Construct L_0^t and L_1^t w.r.t. D;
2: $k := 2$;
3: **while** $L_{k-1}^t \neq \emptyset$ **do begin**
4: $C_k^t = \emptyset$; $L_k^t = \emptyset$;
5: **foreach** $(p_1, p_2) \in L_{k-1}^t \times L_{k-1}^t$ **do begin**
6: $C' :=$ CandidateGen$(p_1, p_2, C_k^t, L_{k-1}^t)$;
7: **forall** $p \in C'$ **do**
8: **if** p is t-frequent w.r.t. D **then** add p to L_k^t;
9: $C_k^t := C_k^t \cup C'$
10: **end**;
11: $k := k + 1$
12: **end**;
13: **return** $\bigcup_{k \geq 0} L_k^t$
end.

Procedure CandidateGen(p_1, p_2, C, L);
Input: block tree patterns p_1, p_2 and sets of block tree patterns C, L;
begin
14: $C' := \emptyset$;
15: **forall** $p_1' \in TB(p_1)$ **do**
16: **forall** $p_2' \in TB(p_2)$ **do**
17: **if** $p_1 \ominus p_1'$ is equivalent to $p_2 \ominus p_2'$ **begin**
18: Let u_1 be the connected point of p_1' to p_1;
19: Let ψ be an isomorphism from $p_1 \ominus p_1'$ to $p_2 \ominus p_2'$;
20: $U := \{u \in V(p_2 \ominus p_2') \mid u$ maps to $\psi(u_1)$ by an automorphism of $p_2 \ominus p_2'\}$;
21: **forall** $u \in U$ **do begin**
22: Let p be a block tree pattern obtained from p_1' and p_2
by identifying u_1 of p_1' with u of p_2;
23: **if** $p \ominus p' \in L$ for all $p' \in TB(p)$ **and** $p \notin C \cup C'$ **then** add p to C'
24: **end**
25: **end**;
26: **return** C'
end;

Fig. 3. An algorithm for generating all frequent block tree patterns with respect to a given set of block trees: We denote by $TB(p)$ the set of all terminal block tree subpatterns of a block tree pattern p

Lemma 5. *Let p_1 and p_2 be two t-frequent $(k-1)$-block tree patterns in L_{k-1}^t for $k \geq 1$. Procedure* CandidateGen *(Fig. 3) computes candidates of t-frequent k-block tree patterns in polynomial time with respect to the numbers of vertices of p_1 and p_2 and the size of L_{k-1}^t.*

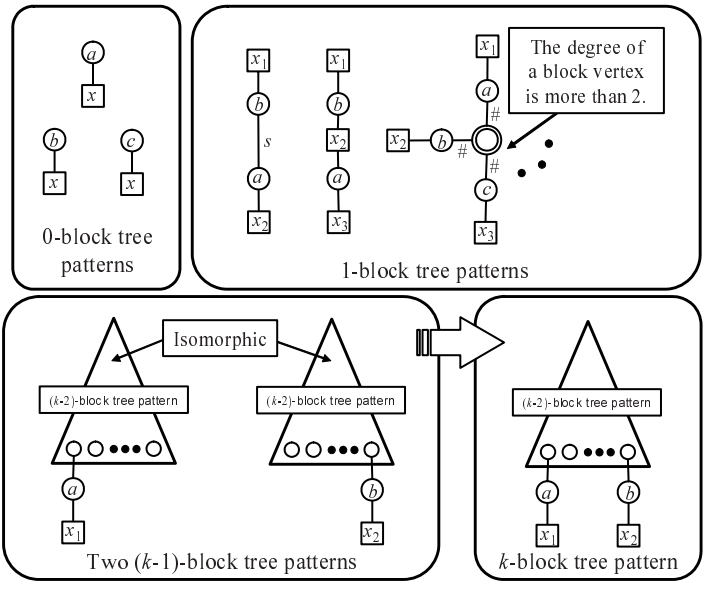


Fig. 4. Examples of 0-block tree patterns and 1-block tree patterns, and a generation of a k-block tree pattern from two $(k-1)$-block tree patterns

Proof. Let N be the size of L^t_{k-1} and $n_{\max}$ the maximum number of vertices of block tree patterns in L^t_{k-1}. Let n be the maximum of the numbers of vertices of p_1 and p_2. The double forall-loop at lines 15 and 16 iterates $O(n^2)$ times. We can decide in $O(n \log n)$ time whether or not the two block tree patterns $p_1 \ominus p'_1$ and $p_2 \ominus p'_2$ are equivalent. The set of vertices U at line 20 can be computed in $O(n)$ time. Since C^t_k contains at most $n^3_{\max} N^2$ block tree patterns, by using a binary search, we can computes the if-statement at line 23 in $O(n^2 \log N + n \log n_{\max})$ time. Since the forall-loop at line 21 iterates $O(n)$ times, it needs $O(n^2(n \log N + \log n_{\max}))$ time totally. Then the time complexity of Procedure CandidateGen is $O(n^4(n \log N + \log n_{\max}))$. □

Lemma 6. *For any $k \geq 2$, L^t_k is correctly computed from L^t_{k-1} in polynomial time with respect to the size of L^t_{k-1} by the above algorithm.*

Proof. The foreach-loop at line 5 of Algorithm FBTPGen computes L^t_k from L^t_{k-1}. From Theorem 1 and Lemma 5, the time complexity of the loop is polynomial time with respect to the size of L^t_{k-1}. □

Finally we have the following theorem.

Theorem 2. *Algorithm* FBTPGen *correctly computes Frequent Block Preserving Outerplanar Graph Pattern Problem.*

		$t = 0.5$			$t = 0.3$			$t = 0.1$		
		C_k^t	L_k^t	time(sec)	C_k^t	L_k^t	time(sec)	C_k^t	L_k^t	time(sec)
	0	10	4	0.03	10	4	0.03	10	5	0.03
	1	53	10	0.1	53	15	0.1	63	24	0.1
	2	32	23	1	72	39	1	150	99	1
	3	159	96	8	254	146	8	544	296	9
	4	338	280	13	663	555	18	1368	1041	20
k	5	797	713	42	1959	1798	72	4819	4128	93
	6	1414	1332	102	5246	4996	276	16338	15010	524
	7	1773	1696	211	12803	12390	1256	51464	49379	4384
	8	1414	1367	309	26556	26330	7431	150050	147461	46995
	9	615	606	222	46826	46677	49593	–	–	–
	10	107	107	57	–	–	–	–	–	–
	11	0	0	3	–	–	–	–	–	–

Fig. 5. Experimental results of Algorithm FBTPGen for generating all t-frequent k-block tree patterns (on Windows XP Professional SP2, JDK 1.5, Pentium D 2.80GHz, 2.00GB RAM)

6 Experimental Result

We have implemented Algorithm FBTPGen (Fig. 3) and tested on a chemical dataset. In our experiments, we used a dataset consisting of 100 outerplanar molecular graphs from the NCI database [4]. The results are given in Fig. 5. We set frequency thresholds to be 0.5, 0.3 and 0.1, and tested on the dataset with respect to the frequencies. The table shows the numbers of candidate and frequent k-block tree patterns, and the runtime in seconds for the generation of L_k^t from L_{k-1}^t. In the table, we only show experimental results obtained by experiments which finished in 50,000 seconds (about 14 hours). The algorithm effectively reduced the amount of candidates for frequent block tree patterns. For example, for the frequency threshold $t = 0.5$, about 92.9% of block tree patterns in $\bigcup_k C_k^t$ are also contained in $\bigcup_k L_k^t$.

In Fig. 6, we give the tables of the number of frequent block tree patterns with respect to the number of bridge variables. For any frequency threshold, the number of frequent block tree patterns with no bridge variable is relatively smaller than the total number of frequent block tree patterns. Our method succeeds in generating many frequent block tree patterns in which some block patterns are connected one another with bridge variables. Then the approach turns out to be useful to find out new frequent patterns in datasets of outerplanar graphs.

The generated patterns certainly contain useless or unimportant patterns from theoretical point of view, because if a frequent block tree pattern has a labeled edge, a block tree pattern which is obtained from the frequent block tree pattern by replacing the labeled edge with a bridge variable is also frequent. In order to generate more refined frequent block tree patterns and bpo-graph patterns, we are developing mining methods such as maximal frequent or closed frequent pattern mining.

	The number of bridge variables ($t = 0.5$)												
	0	1	2	3	4	5	6	7	8	9	10	11	Total
0	4												4
1	3	7											10
2	5	11	7										23
3	11	35	37	13									96
4	14	71	112	67	16								280
k 5	14	101	241	236	105	16							713
6	5	79	291	453	355	133	16						1332
7	0	31	178	448	548	365	116	10					1696
8	0	5	48	189	389	417	251	66	2				1367
9	0	0	4	25	88	172	183	108	26	0			606
10	0	0	0	0	2	13	31	35	21	5	0		107
11	0	0	0	0	0	0	0	0	0	0	0	0	0

	The number of bridge variables ($t = 0.3$)										
	0	1	2	3	4	5	6	7	8	9	Total
0	4										4
1	6	9									15
2	5	20	14								39
3	15	55	58	18							146
k 4	26	143	220	137	29						555
5	34	285	621	567	246	45					1798
6	26	422	1309	1703	1118	365	53				4996
7	11	522	2192	3859	3519	1780	459	48			12390
8	2	537	2971	6676	7977	5509	2192	435	31		26330
9	0	466	3182	8953	13395	11895	6446	2031	297	12	46677

	The number of bridge variables ($t = 0.1$)									
	0	1	2	3	4	5	6	7	8	Total
0	5									5
1	12	12								24
2	22	50	27							99
3	41	114	106	35						296
k 4	61	295	405	229	51					1041
5	122	687	1394	1281	548	96				4128
6	208	1451	3758	4873	3360	1181	179			15010
7	295	2718	8717	14336	13498	7352	2177	286		49379
8	426	4540	17727	35498	41824	30395	13398	3297	356	147461

Fig. 6. The numbers of t-frequent k-block tree patterns with r bridge variables for $r \geq 0$

7 Conclusion and Future Works

In this paper, we have considered a data mining problem of extracting structural features from semi-structured data whose data can be expressed by outerplanar graphs. Firstly, we have defined a block preserving outerplanar graph pattern

as a new graph pattern having an outerplanar graph structure and structured variables. Secondly, we have presented an incremental polynomial time Apriori-like algorithm for enumerating all frequent bpo-graph patterns with respect to a given finite set of outerplanar graphs. Finally, by reporting experimental results on a subset of the NCI dataset, we have evaluated the performance of our algorithm. Currently we are developing learning algorithms for the class of bpo-graph patterns in the framework of computational learning theory.

In [3], we introduced unordered term trees, which are unordered tree patterns with internal structured variables, and showed that a matching problem for the class of unordered term trees with variables of more than 3 ports is NP-complete. From this, we easily observe that the matching problem for the class of bpo-graph patterns with variables of more than 3 ports is also NP-complete. It is an open problem whether or not there is a polynomial time matching algorithm for the class of bpo-graph patterns with variables of at most 3 ports.

References

1. Horváth, T., Ramon, J., Wrobel, S.: Frequent Subgraph Mining in Outerplanar Graphs. In: Proc. KDD 2006, pp. 197–206 (2006)
2. Lingas, A.: Subgraph isomorphism for biconnected outerplanar graphs in cubic time. Theoretical Computer Science 63, 295–302 (1989)
3. Miyahara, T., Shoudai, T., Uchida, T., Takahashi, K., Ueda, H.: Polynomial time matching algorithms for tree-like structured patterns in knowledge discovery. In: Terano, T., Chen, A.L.P. (eds.) PAKDD 2000. LNCS, vol. 1805, pp. 5–16. Springer, Heidelberg (2000)
4. National Cancer Institute - Chemical Dataset, http://cactus.nci.nih.gov/
5. Shamir, R., Tsur, D.: Faster subtree isomorphism. Journal of Algorithms 33(2), 267–280 (1999)
6. Suzuki, Y., Inomae, K., Shoudai, T., Miyahara, T., Uchida, T.: A polynomial time matching algorithm of structured ordered tree patterns for data mining from semistructured data. In: Matwin, S., Sammut, C. (eds.) ILP 2002. LNCS (LNAI), vol. 2583, Springer, Heidelberg (2003)
7. Suzuki, Y., Shoudai, T., Uchida, T., Miyahara, T.: Ordered term tree languages which are polynomial time inductively inferable from positive data. Theoretical Computer Science 350, 63–90 (2006)
8. Takami, R., Suzuki, Y., Uchida, T., Shoudai, T., Nakamura, Y.: Polynomial time inductive inference of TTSP graph languages from positive data. In: Kramer, S., Pfahringer, B. (eds.) ILP 2005. LNCS (LNAI), vol. 3625, pp. 366–383. Springer, Heidelberg (2005)
9. Uchida, T., Shoudai, T., Miyano, S.: Parallel algorithm for refutation tree problem on formal graph systems. IEICE Transactions on Information and Systems E78-D(2), 99–112 (1995)
10. Yamasaki, H., Shoudai, T.: A polynomial time algorithm for finding linear interval graph patterns. In: Cai, J.-Y., Cooper, S.B., Zhu, H. (eds.) TAMC 2007. LNCS, vol. 4484, pp. 67–78. Springer, Heidelberg (2007)

Relational Macros for Transfer in Reinforcement Learning

Lisa Torrey[1], Jude Shavlik[1],
Trevor Walker[1], and Richard Maclin[2]

[1] University of Wisconsin, Madison WI 53706, USA
[2] University of Minnesota, Duluth, MN 55812, USA

Abstract. We describe an application of inductive logic programming to transfer learning. Transfer learning is the use of knowledge learned in a source task to improve learning in a related target task. The tasks we work with are in reinforcement-learning domains. Our approach transfers relational macros, which are finite-state machines in which the transition conditions and the node actions are represented by first-order logical clauses. We use inductive logic programming to learn a macro that characterizes successful behavior in the source task, and then use the macro for decision-making in the early learning stages of the target task. Through experiments in the RoboCup simulated soccer domain, we show that Relational Macro Transfer via Demonstration (RMT-D) from a source task can provide a substantial head start in the target task.

1 Introduction

Knowledge transfer is an inherent aspect of human learning. When we learn to perform a task, we rarely start from scratch; instead we recall relevant knowledge from previous learning experiences and apply it. Transferring knowledge this way helps us to master new tasks more quickly.[1]

Machine learning techniques are often designed to address isolated learning tasks. However, many machine learning domains contain multiple related tasks. *Transfer learning* approaches take advantage of these relationships, using knowledge learned in a *source task* to speed up learning in a related *target task*. Algorithms that allow successful transfer represent progress towards making machine learning as effective as human learning.

One area in which transfer is often desirable is *reinforcement learning* (RL), since standard RL algorithms can require long training times. The RL domain that we use in this work is the simulated soccer project RoboCup [9]. In Section 2 we give an overview of RL and the RoboCup domain.

Several algorithms for transfer in domains like RoboCup have been proposed, some of which we discuss in Section 3. In our own recent work [20], we introduce an approach that transfers skills using inductive logic programming (ILP), where a *skill* is a type of action that the RL agent uses. In this paper, we extend that approach by transferring *strategies*, which are action plans that may require

[1] Appears in the 17th Conference on Inductive Logic Programming, 2007.

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 254–268, 2008.

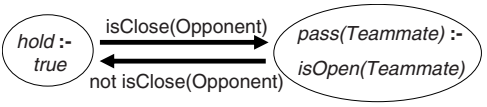


Fig. 1. A possible strategy for the RoboCup game KeepAway [13], in which the RL agent in possession of the soccer ball must execute a series of *hold* or *pass* actions to prevent its opponents from getting the ball. The rules inside nodes show how to choose actions. The labels on arcs show the conditions for taking transitions. Each node has an implied self-transition that applies by default if no exiting arc applies.

composing several skills. We continue to use ILP to learn strategies, and we represent them with a structure that we call a *relational macro*.

A relational macro is a finite-state machine (FSM) that uses first-order logic for decision-making. An FSM is a behavior model consisting of a set of nodes and transitions. To use a macro, an RL agent takes transitions to move between nodes representing internal states, and it chooses actions to take in each node. Its choices are determined by first-order logical clauses. Figure 1 shows a simple example of a relational macro and Section 4 provides more details on how a macro is executed.

We use inductive logic programming (ILP) to learn macros because domains like RoboCup are inherently relational. To our knowledge, fully relational RL approaches have not yet been successfully applied in domains as complex as RoboCup. However, as we showed with skill transfer, relational information can be successfully transferred between RoboCup tasks. Therefore we continue to use ILP in this approach, describing source-task behavior and relational macros in first-order logic.

Relational-macro transfer begins by examining existing source-task episodes and analyzing them to learn a successful strategy in the form of a macro. Section 5 describes our algorithm for this learning stage. There are several possible ways to use the macro to improve learning in the target task; we use it to demonstrate the successful strategy, as described in Section 6. After a short demonstration period that gives the target-task learner a head start, we continue learning the task with standard RL. We call this approach Relational Macro Transfer via Demonstration (RMT-D).

2 Reinforcement Learning in RoboCup

In reinforcement learning [16], an agent navigates through an environment trying to earn rewards or avoid penalties. The environment's state is described by a set of features, and the agent takes actions to cause the state to change. In one common form of RL called Q-learning [22], the agent learns a Q-function to estimate the value of taking an action from a state. An agent's *policy* is typically to take the action with the highest Q-value in the current state, except for occasional exploratory actions. After taking the action and receiving some reward (possibly zero), the agent updates its Q-value estimates for the current state.

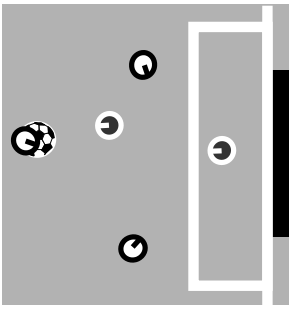


Fig. 2. Snapshot of a 3-on-2 BreakAway game. The attacking players have possession of the ball and are maneuvering against the defending team towards the goal.

Stone and Sutton [13] introduced RoboCup as an RL domain that is challenging because of its large, continuous state space and non-deterministic action effects. The RoboCup project [9] has the overall goal of producing robotic soccer teams that compete on the human level, but it also has a software simulator for research purposes. Since the full game of soccer is quite complex, researchers have developed several simpler games within the RoboCup simulator. See Figure 2 for a snapshot of one of these games.

In M-on-N BreakAway [21], the objective of the M reinforcement learners called *attackers* is to score a goal against $N-1$ hand-coded *defenders* and a *goalie*. The game ends when they succeed, when an opponent takes the ball, when the ball goes out of bounds, or after a time limit of 10 seconds. The learners receive a +1 reward if they score a goal and 0 reward otherwise. The attacker who has the ball may choose to move (ahead, away, left, or right with respect to the goal center), pass to a teammate, or shoot (at the left, right, or center part of the goal).

RoboCup tasks are inherently multi-agent games, but a standard simplification is to have only one learning agent. This agent controls the attacker currently in possession of the ball, switching its "consciousness" between attackers as the ball is passed. Attackers without the ball follow simple hand-coded policies that position them to receive passes.

Table 1 shows the state features for BreakAway, which mainly consist of distances and angles between players and the goal. They are represented in logical notation, though our RL algorithm uses the grounded versions of these predicates in a fixed-length feature vector. Capitalized atoms indicate typed variables, while constants and predicates are uncapitalized. The attackers (labeled *a0*, *a1*, etc.) are ordered by their distance to the agent in possession of the ball (*a0*), as are the non-goalie defenders (*d0*, *d1*, etc.).

Our RL implementation uses a $SARSA(\lambda)$ variant of Q-learning [15] and employs a support vector machine for function approximation [7]. We relearn the Q-function using the SVM after every batch of 25 games. The exploration rate begins at 2.5% and decays exponentially over time. Stone and Sutton [13] found that discretizing the continuous features into Boolean interval features

Table 1. The features that describe a BreakAway state

distBetween(a0, Player)	distBetween(Attacker, goalCenter)
distBetween(Attacker, ClosestDefender)	distBetween(a0, GoalPart)
angleDefinedBy(Attacker, a0, ClosestDefender)	timeLeft
angleDefinedBy(topRight, goalCenter, a0)	distBetween(Attacker, goalie)
angleDefinedBy(GoalPart, a0, goalie)	angleDefinedBy(Attacker, a0, goalie)

called *tiles* is useful for learning in RoboCup; following this approach we create 32 tiles per feature.

Agents in the games of 2-on-1, 3-on-2, and 4-on-3 BreakAway take approximately 3000 training episodes to reach a performance asymptote in our system. These three games are similar, but their differences in the numbers of attackers and defenders cause substantial differences in their optimal policies. The largest difference is between 2-on-1 and the others, since there are no non-goalie defenders in 2-on-1 BreakAway. Despite the differences, the tasks do have the same objective and can be expected to require similar strategies, which makes relational macros an attractive technique for transferring between them.

3 Related Work in Transfer Learning

The goal in transfer learning is to speed up learning in a target task by transferring knowledge from a related source task. One straightforward way to do this in reinforcement learning is to begin performing the target task using the learned source-task models. Taylor et al. [19] use this type of transfer method, which we refer to as *model reuse*.

Another approach that has been proposed is to follow source-task policies during the exploration steps of normal RL in the target task, instead of doing random exploration. This approach is referred to as *policy reuse* and is performed by Fernandez and Veloso [5].

Our previous work includes a method called *skill transfer* [20]. In skill transfer, we learn rules with ILP that indicate when the agent chooses to take a single source-task action. There are multiple ways that these *skills* could be used in the target task; we use an advice-taking approach in this previous work. Our *advice* places soft constraints on the target-task solution that can be followed or ignored according to how successful they are. Taylor and Stone [18] also learn a set of rules for taking actions, and they propose different advice-taking mechanisms: for example, they give a Q-value bonus to the advised action.

There are also approaches for transferring multi-step action sequences, such as those of Perkins and Precup [10] and Soni and Singh [11]. Known as *options*, these sequences have their own internal Q-functions that are followed until they reach a termination state. The target-task learner treats options as alternative actions. Croonenborghs et al. [1] learn relational options for use in relational

reinforcement learning (RRL). Options are often used in hierarchical RL [2] as well as in transfer learning.

Relational reinforcement learning [17] itself is a related topic. In RRL, state descriptions and learned models use first-order logic, which clearly provides opportunities for transferring concepts in first-order logic. Driessens et al. [4] and Stracuzzi and Asgharbeygi [14] point out some of these opportunities.

We propose to perform transfer by learning relational macros and using them to demonstrate successful behavior in the target task. Our approach is related to several of the methods described above. It could be viewed as a type of model reuse or policy reuse that creates an abstract version of the source-task model instead of reusing it directly. Like skill transfer it uses ILP, but it involves multi-step strategies instead of single actions. It shares the idea of transferring sub-policies with option transfer, but an option traditionally represents a single policy while a macro contains a separate sub-policy at each node.

4 Executing a Relational Macro

We have defined a relational macro as a finite-state machine [6]. An FSM models the behavior of a system in the form of a directed graph. The nodes of the graph represent states of the system, and in our case they represent internal states of the agent in which different policies apply.

The policy of a node can be to take a single action, such as *move(ahead)* or *shoot(goalLeft)*, or to choose from a class of actions, such as *pass(Teammate)*. In the latter case a node has first-order logical clauses to decide which grounded action to choose. An FSM begins in a start node and has conditions for transitioning between nodes. In a relational macro, these conditions are also sets of first-order logical clauses.

We refer again to the example macro in Figure 1. When executing this macro, a KeepAway agent begins in the initial node on the left. The only action it can choose in this node is *hold*. It remains there, taking the default self-transition, until the condition *isClose(Opponent)* becomes true for some opponent player. Then it transitions to the second node, where it evaluates the *pass(Teammate)* rule to choose an action. If the rule is true for just one teammate player, it passes to that teammate; if several teammates qualify, it randomly chooses between them; if no teammate qualifies, it abandons the macro and reverts to using the Q-function to choose actions. The receiving teammate then becomes the learning agent, and it remains in the *pass* node if an opponent is close or transitions back to the *hold* node otherwise.

Figure 1 is a simplification in one respect: each transition and node in a macro has an entire set of rules, rather than just one rule. This allows us to represent disjunctive conditions. When more than one grounded action or transition is possible (when multiple rules match), the agent obeys the rule that has the highest score. The score of a rule is the estimated probability that following it will lead to a successful game. We estimate these probability scores from source-task data.

5 Learning a Relational Macro

We learn a macro by analyzing source-task data. We assume that this data is available because we have previously learned the source task and stored the games generated during the learning process. The method by which the source task was learned is not particularly important, since the data we use only consists of states, actions, and rewards. However, it is important that the data include source-task games from early in the learning curve as well as later, so that there are examples of games that do not attempt to use the final learned strategy. In our system we include all 3000 games from the source-task learning curve.

Given this data, we use inductive logic programming (ILP) to characterize successful behavior in the source task. Specifically, we use a locally modified version of Aleph [12]. The Aleph algorithm selects an example, builds the most specific clause that entails the example within the provided language restrictions, and searches for a more general clause that maximizes a provided scoring function.

The *precision* of a rule is the fraction of examples it calls positive that are truly positive, and the *recall* is the fraction of truly positive examples that it correctly calls positive. The scoring function we use is

$$F(1) = \frac{2 * Precision * Recall}{Precision + Recall}$$

because we consider both precision and recall to be important. We use both the heuristic and randomized search algorithms provided by Aleph.

Recall that a macro consists of a set of nodes along with rulesets for transitions and action choices. The simplest algorithm for learning a macro might be to provide Aleph with language restrictions that allow it to learn both the structure and the rulesets simultaneously. However, this would be a very large search space. To make the search more feasible, we separate it into two phases: first we learn the structure, and then we learn each ruleset independently. Each phase therefore has its own language restrictions, which we detail in the following sections. The overall algorithm is summarized in Table 2.

Note that one final step might be necessary if the actions and features in the source and target tasks are not identically named: a *mapping* from source-task names to target-task names, as in Torrey et al. [20,21]. Our approach does not even require the tasks to be completely isomorphic, because we can set the Aleph language restrictions so that only source-task elements that have corresponding target-task elements appear in the macro.

5.1 Structure Learning

The first phase in our RMT-D algorithm for learning a macro is the structure-learning phase. The objective is to find a sequence of actions that distinguishes successful games from unsuccessful games. The sequence does not need to separate the games perfectly, and indeed we should not expect it to, because it does not yet have any conditions on states. The structure only needs to provide a good starting point for the second phase.

Table 2. Our RMT-D algorithm for learning a relational macro from a source task

Phase 1: Structure learning
- Collect games from source task
- Let *Pos* = high-reward games
- Let *Neg* = low-reward games
- Learn a macro sequence that distinguishes *Pos* from *Neg*

Phase 2: Ruleset learning
- Collect games G_{good} that contain the macro sequence and are high-reward
- Collect games G_{bad} that are low-reward
- For each node N in the macro sequence
  - For each action A represented by node N
    - Let *Pos* = G_{good} states from node N that took action A
    - Let *Neg* = G_{good} states from node N that took action B $\neq$ A $\cup$ G_{bad} states that ended with action A
    - Learn a ruleset that distinguishes *Pos* from *Neg*
- For each transition T in the macro
  - Let *Pos* = G_{good} states that took transition T
  - Let *Neg* = G_{good} states that could have taken transition T and did not $\cup$ G_{bad} states that ended with transition T
  - Learn a ruleset that distinguishes *Pos* from *Neg*

The language restrictions for Aleph in this phase are as follows. Let the predicate *actionTaken(G, S_1, A, P, S_2)* denote that action A with argument P was taken in game G at step S_1 and repeated until step S_2. Aleph must construct a clause *macroSequence(G)* with a body that contains a combination of these predicates. The first predicate may introduce two new variables, S_1 and S_2, but the rest must use an existing variable for S_1 while introducing another new variable S_2. In this way Aleph finds a connected sequence of actions that translates directly to a linear node structure.

We provide Aleph with sets of positive and negative examples, where positives are games with high overall reward and negatives are those with low overall reward. For BreakAway, this is a straightforward separation of scoring and non-scoring games. For tasks with more continuous rewards, we could set thresholds or upper and lower percentiles on the overall reward acquired during a game.

We store all the clauses that Aleph encounters during its search that separate the positive and negative examples with at least 50% accuracy. After the heuristic and randomized searches finish, we take the sequence with the highest F(1) score as the macro structure.

For instance, suppose that the scoring BreakAway games consistently look like these examples:

Game 1: move(ahead), pass(a1), shoot(goalRight)
Game 2: move(ahead), move(ahead), pass(a2), shoot(goalLeft)

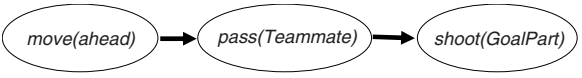


Fig. 3. The structure that corresponds to the example macro clause in Section 5.1

Assuming that the non-scoring games have different patterns than the examples above do, Aleph might learn the following clause to characterize a scoring game:

```
macroSequence(Game) :-
    actionTaken(Game, StateA, move, ahead, StateB),
    actionTaken(Game, StateB, pass, Teammate, StateC),
    actionTaken(Game, StateC, shoot, GoalPart, gameEnd).
```

The macro structure corresponding to this sequence is shown in Figure 3. The policy in the first node will be to take a single action, *move(ahead)*. In the second node the policy will be to consider multiple *pass* actions, and in the third node the policy will be to consider multiple *shoot* actions. The conditions for choosing an action, and for taking transitions between nodes, are learned in the next phase.

5.2 Ruleset Learning

The second phase in our RMT-D algorithm for learning a macro is the ruleset-learning phase. The objective is to describe when transitions and actions should be taken based on the RL state features. We learn a ruleset for each transition and each action independently, so that we perform several smaller, in-depth seaches rather than one large search. Because of this, variables in these rules are local to a node rather than global to the entire macro.

The language restrictions for Aleph in this phase are as follows. There is one predicate for each state feature of the RL task (for BreakAway, these are in Table 1). To describe the conditions on state S under which a transition should be taken, Aleph must construct a clause *transition(S)* with a body that contains a combination of these predicates. To describe the conditions under which an action should be taken, Aleph must construct a clause *action(S, Action, ActionArg)*.

Aleph may learn some action rules in which the action argument is grounded, such as *action(S, move, ahead)*, as well as rules in which the action argument remains a variable, such as *action(S, pass, Teammate)*. In the case of the *move* action in BreakAway the action argument in a rule is always grounded, since the original state features do not include useful references to move directions. We could have defined additional predicates that did, but we chose to use only the original features. Note that it is still possible to have a state *move(Direction)* for taking multiple move actions, but the rules for choosing a grounded move action will use only grounded arguments.

We provide Aleph with sets of positive and negative examples, consisting of states in source-task games that took the transition or action. Consider the macro structure in Figure 3; we will describe the action datasets for the *pass* node and

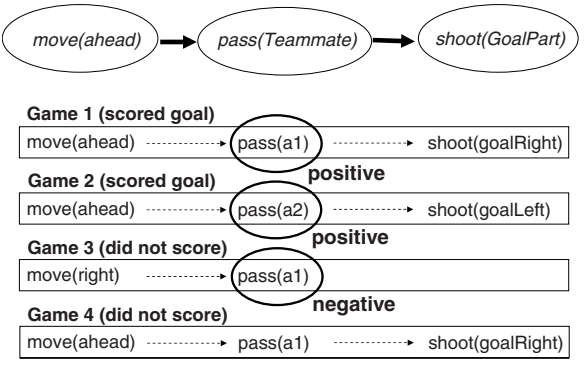


Fig. 4. Training examples (states circled) for *pass(Teammate)* rules in the second node of the pictured macro. The pass states in Games 1 and 2 are positive examples. The pass state in Game 3 is a negative example; this game did not follow the macro, but the pass action led directly to a negative game outcome. The pass state in Game 4 is not an unambiguous example because a later action may have been responsible for the bad outcome.

the transition datasets for the transition from the *move* node to the *pass* node. Let G_{good} represent the set of high-reward source-task games that contain the macro sequence and let G_{bad} represent the set of low-reward source-task games.

In the action datasets for the *pass* node, the positive examples are states in G_{good} games that fall into that node. The negative examples are states in G_{bad} games in which the last step of the unsuccessful game was the node action, *pass*. This indicates that the *pass* action led directly to a negative game outcome. Figure 4 illustrates some hypothetical action-choice examples.

In the transition datasets for the transition from the *move* node to the *pass* node, the positive examples are states in G_{good} games that match the *pass* node and for which the previous state matched the *move* node. A negative example is a state in a G_{good} game that does not match the *pass* node even though the previous state matched the *move* node. Other negative examples are states in G_{bad} games in which the last step of the unsuccessful game was a transition from the *move* node to the *pass* node. Figure 5 illustrates some hypothetical transition examples.

As in the first phase, we store all the clauses that Aleph encounters during the search that classify the training data with at least 50% accuracy. However, instead of selecting a single best clause as we did in the previous phase, we collect from these a ruleset for each transition and each action. We wish to have one strategy (i.e. one finite-state machine), but there may be multiple reasons for making internal choices.

Our procedure for greedily selecting which clauses are included in a ruleset is summarized in Table 3. We sort the rules by decreasing precision and walk through the list, adding rules to the final ruleset if they increase the set's recall and do not decrease its F(1) score.

We assign each rule a score that may be used to decide which rule to obey if multiple rules match while executing the macro. The score is an estimate

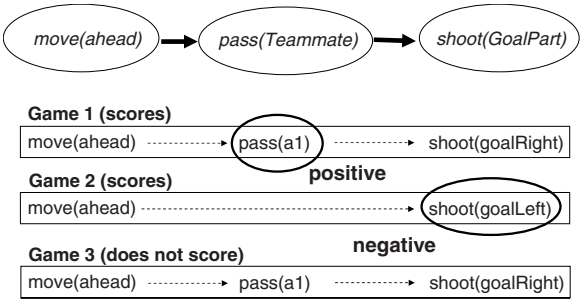


Fig. 5. Training examples (states circled) for the transition from *move* to *pass* in the pictured macro. The pass state in Game 1 is a positive example. The shoot state in Game 2 is a negative example; the game began by following the macro but did not take the transition from *move* to *pass*. The pass state in Game 3 is not an unambiguous example because a later step may have been responsible for the bad outcome.

of the probability that following the rule will lead to a successful game. We determine this estimate by collecting training-set games that followed the rule and calculating the fraction of these that ended successfully.

6 Transferring a Relational Macro

A relational macro describes a strategy that was successful in the source task. There are several ways we could use this information to improve learning in a related target task. One possibility is to treat it as *advice* [20], as we did in skill transfer, putting soft constraints on the Q-learner that influence its solution. The benefit of this approach is its robustness to error: if the source-task knowledge is less appropriate to the target task than the user expected, the target-task agent can learn to disregard the soft constraints, avoiding negative transfer effects.

On the other hand, the advice-taking approach is conservative and can be somewhat slow to reach its full effect, even when the source-task knowledge is highly appropriate to the target task. Since a macro is a full strategy rather than isolated skills, we might achieve good target-task performance more quickly by *demonstrating* the strategy in the target task and using it as a starting point for learning. This is a more aggressive approach, carrying more risk for negative transfer if the source and target tasks are not similar enough. Still, if the user believes that the tasks are similar enough, the potential benefits could outweigh that risk.

There are intermediate approaches with more moderate benefits and risks, such as using the macro as an option. Here we take full advantage of the potential benefits, and also provide a contrasting method to skill transfer, by presenting the more aggressive demonstration method.

Our target-task learner therefore begins by simply executing the macro strategy for a set of episodes, instead of exploring randomly as an untrained RL agent would traditionally do. In this demonstration period, we generate examples of Q-values: each time the macro chooses an action because a high-scoring rule

Table 3. The RMT-D procedure for selecting the final ruleset for one transition or action. Rules are added to the final set if they cover previously uncovered positive examples and do not decrease the overall score. The scoring function is the F(1) measure.

Let R = all rules encountered with $> 50\%$ accuracy
$S = R$ sorted by decreasing precision on the training set
$T = \emptyset$
For each rule $r \in S$
    $U = T \cup \{r\}$
    If recall(U) $>$ recall(T) and score(U) $\geq$ score(T)
    Then $T = U$
Return *FinalRuleset* $= T$

---

matched, we use the rule score to estimate the Q-value of the action. Recall the the rule score is the estimated probability that following the rule leads to a successful game. Since BreakAway has Q-values ranging from zero to one, we simply set the estimate equal to the rule score (if this were not the case, we could multiply the probability by an appropriate scaling factor to fit a larger Q-value range). We also use rule scores to produce Q-value estimates for other actions for which rules fired. Finally, we infer that actions for which no rules fired had very low Q-values, which in the BreakAway domain we estimate as zero.

Note that the examples with low estimated Q-values are necessary to ensure that the initial Q-function is not overly optimistic in unexplored areas. Driessens and Dzeroski also encountered this problem in their work on guidance in RRL; they addressed it by interleaving imitation with exploration [3].

For each step of the demonstration we therefore have a Q-value estimate for each action, and via support vector regression we use these to learn an initial Q-function for the target task. The demonstration period lasts for 100 games in our system, and as usual after each batch of 25 games we relearn the Q-function. After 100 games, we continue learning the target task with standard RL. This generates new Q-value examples in the standard way, and we combine these with the old macro-generated examples as we continue relearning the Q-function after each batch. As the new examples accumulate, we gradually drop the old examples by randomly removing them at the rate that new ones are being added.

Since standard RL has to act mostly randomly in the early steps of a task, a good macro strategy can provide a large immediate advantage. The performance level of the demonstrated strategy is unlikely to be as high as the target-task agent can achieve with further training, unless the tasks are similar enough to make transfer a trivial problem, but the hope is that the learner can smoothly improve its performance from the level of the demonstration up to its asymptote. If there is limited time and the target task cannot be trained to its asymptote, then the immediate advantage that macros can provide may be even more valuable in comparison to methods like skill transfer.

7 Experimental Results

We present results from transfer experiments with RMT-D in the RoboCup domain. To test our approach, we learn a macro from data acquired while training 2-on-1 BreakAway and transfer it to both 3-on-2 and 4-on-3 BreakAway. We learn the source task with standard RL for 3000 games, and then we train the target tasks for 3000 games to show both the initial advantage of the macros and the behavior as training continues.

The macros that RMT-D learned from the five source runs all had similar structures. The most common version is shown in Figure 6. In one of the runs the initial *pass* node was not included, and the ordering of *shoot(goalRight)* and *shoot(goalLeft)* varied, as would be expected in the symmetrical BreakAway domain. The presence of two *shoot* nodes may seem counterintuitive, but it appears that the RL agent uses the first shot as a feint to lure the goalie in one direction, counting on a teammate to intercept the shot before it reaches the goal. When it does, the learning agent switches to the teammate in possession of the ball and performs the second shot, which is actually intended to score. This tendency of RL agents to use actions in unintended ways is an indication of the difficulties that can arise when learning relational concepts from RL data.

Figures 7 and 8 show our results in 3-on-2 and 4-on-3 BreakAway respectively. We compare our approach against Q-learning as well as two related transfer methods: model reuse [19] and skill transfer [20]. Each curve in the figure is an average of 25 runs and has points smoothed over the previous 500 games to smooth over the high variance in the RoboCup domain. For the transfer algorithms, there are five target-task runs generated from each of five source-task runs, to allow for variance in both stages of learning.

Our agents in 2-on-1 BreakAway reach a performance asymptote of scoring in approximately 70% of the episodes. The macros learned from the 2-on-1 source runs, when executed in 2-on-1 BreakAway, score in approximately 50% of the episodes. (A random policy scores in less than 1% of the episodes.) The macros therefore appear to capture the majority of the successful behavior of the source task, though they do not describe it completely. Capturing source-task behavior more completely, while avoiding overfitting, is one topic for future work.

All of the transfer algorithms speed up learning in comparison to Q-learning, but the benefits they provide are different. Model reuse and relational macros both provide an advantage in the early performance of the target-task learner. RMT-D produces a larger advantage in these scenarios than model reuse does,

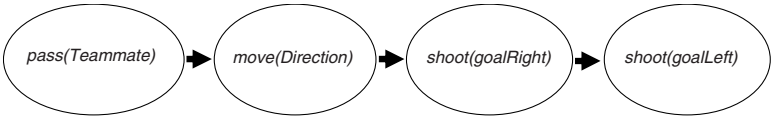


Fig. 6. One of the five macro structures learned from 2-on-1 BreakAway runs. There are between 10 and 20 rules associated with each transition and action, so those are not shown.

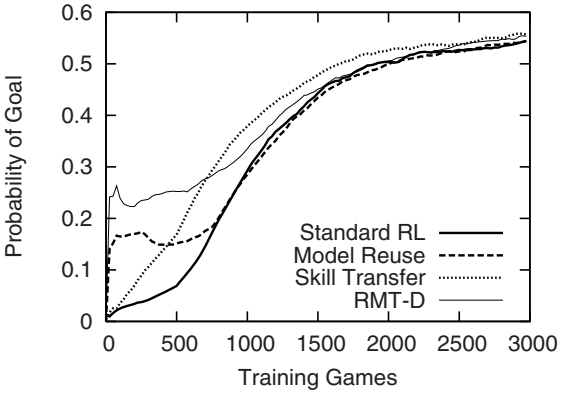


Fig. 7. Probability of scoring a goal in 3-on-2 BreakAway, with Q-learning and with three transfer approaches that use 2-on-1 BreakAway as the source task

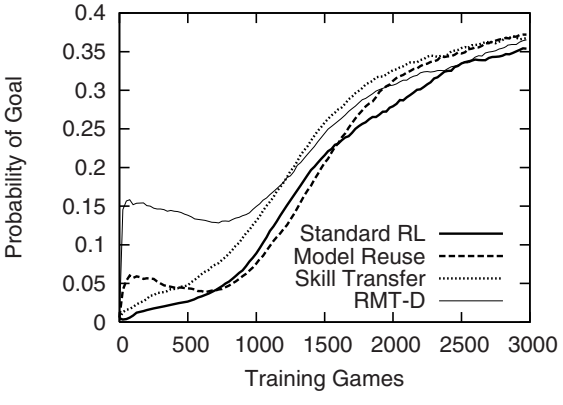


Fig. 8. Probability of scoring a goal in 4-on-3 BreakAway, with Q-learning and with three transfer approaches that use 2-on-1 BreakAway as the source task

and it scales better as the distance between the source and target grows. Skill transfer provides no initial benefit, but then develops a steady advantage over Q-learning. During the middle section of the learning curve it performs slightly better than RMT-D before they all converge at the asymptote.

In pointwise t-test comparisons at the 99% confidence level, the RMT-D curve is significantly above the model-reuse curve for the first 1100 episodes in Figure 7 and 1425 episodes in Figure 8. The RMT-D curve is significantly above the skill-transfer curve for the first 575 episodes in Figure 7 and 875 episodes in Figure 8. The skill-transfer curve is significantly above the RMT-D curve at just one point in Figure 7 (at 1825 episodes) and never in Figure 8, and the model-reuse curve is never significantly above the RMT-D curve in either figure.

We also tried an algorithm that combines skill transfer via advice with RMT-D. The combination is straightforward: we begin by demonstrating the macro as in

RMT-D, and we incorporate advice when learning the Q-function as in skill transfer. This produces a learning curve (not shown) that is not significantly different from the RMT-D curve. The substantial early effects of transferring a macro via demonstration apparently overwhelm the effects of skill-transfer advice.

8 Conclusions and Future Work

Knowledge transfer in reinforcement learning is an interesting and challenging problem, and inductive logic programming is a powerful tool to apply to it. The use of ILP allows us to transfer the kind of information that humans might transfer: strategies with decisions in first-order logic. We describe an approach for transferring relational macros from a source task that gives the target-task learner a significant head start. Our approach produces consistently higher initial performance than standard RL and several related transfer methods.

In future work, we plan to investigate alternative macro designs that may capture the source-task behavior more completely. While a single linear action sequence appears to explain the majority of our agents' success in the source task, other configurations might perform better. We are interested in trying a statistical relational learning (SRL) approach to estimate probabilities and to make decisions from rulesets.

Our RMT-D algorithm is most effective when the user is confident that the source-task strategy is a reasonable approximation of a good target-task strategy. However, relational macros might be applicable in more distant transfer scenarios, such as when only part of a source-task strategy is useful in a target task. We plan to investigate alternative ways to apply relational macros in the target task to make this possible. Potential frameworks for this include options [1] and advice-taking [8]. We are also interested in incorporating human advice into relational structures.

Another direction for future work is the refinement of relational macros during target-task learning. The parameters or structure of a macro could be updated based on early experience in the target task. This is a problem of *theory refinement*, which is an area of interest for transfer learning.

Acknowledgements

This research is supported by DARPA IPTO grants HR0011-04-1-0007 and FA8650-06-C-7606.

References

1. Croonenborghs, T., Driessens, K., Bruynooghe, M.: Learning relational skills for inductive transfer in relational reinforcement learning. In: International Conference on Inductive Logic Programming (2007)
2. Dietterich, T.: Hierarchical reinforcement learning with the MAXQ value function decomposition. Journal of Artificial Intelligence Research 13, 227–303 (2000)

3. Driessens, K., Dzeroski, S.: Integrating guidance into relational reinforcement learning. Machine Learning 57(3), 271–304 (2004)
4. Driessens, K., Ramon, J., Croonenborghs, T.: Transfer learning for reinforcement learning through goal and policy parametrization. In: ICML Workshop on Structural Knowledge Transfer for Machine Learning (2006)
5. Fernandez, F., Veloso, M.: Policy reuse for transfer learning across tasks with different state and action spaces. In: ICML Workshop on Structural Knowledge Transfer for Machine Learning (2006)
6. Gill, A.: Introduction to the Theory of Finite-state Machines. McGraw-Hill, New York (1962)
7. Maclin, R., Shavlik, J., Torrey, L., Walker, T.: Knowledge-based support vector regression for reinforcement learning. In: IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games (2005)
8. Maclin, R., Shavlik, J., Torrey, L., Walker, T., Wild, E.: Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In: AAAI Conference on Artificial Intelligence (2005)
9. Noda, I., Matsubara, H., Hiraki, K., Frank, I.: Soccer server: A tool for research on multiagent systems. Applied Artificial Intelligence 12, 233–250 (1998)
10. Perkins, T., Precup, D.: Using options for knowledge transfer in reinforcement learning. Technical Report UM-CS-1999-034 (1999)
11. Soni, V., Singh, S.: Using homomorphisms to transfer options across continuous reinforcement learning domains. In: AAAI Conference on Artificial Intelligence (2006)
12. Srinivasan, A.: The Aleph manual (2001)
13. Stone, P., Sutton, R.: Scaling reinforcement learning toward RoboCup soccer. In: International Conference on Machine Learning (2001)
14. Stracuzzi, D., Asgharbeygi, N.: Transfer of knowledge structures with relational temporal difference learning. In: ICML Workshop on Structural Knowledge Transfer for Machine Learning (2006)
15. Sutton, R.: Learning to predict by the methods of temporal differences. Machine Learning 3, 9–44 (1988)
16. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
17. Tadepalli, P., Givan, R., Driessens, K.: Relational reinforcement learning: An overview. In: ICML Workshop on Relational Reinforcement Learning (2004)
18. Taylor, M., Stone, P.: Cross-domain transfer for reinforcement learning. In: International Conference on Machine Learning (2007)
19. Taylor, M., Stone, P., Liu, Y.: Value functions for RL-based behavior transfer: A comparative study. In: AAAI Conference on Artificial Intelligence (2005)
20. 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, Springer, Heidelberg (2006)
21. Torrey, L., Walker, T., Shavlik, J., Maclin, R.: Using advice to transfer knowledge acquired in one reinforcement learning task to another. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, Springer, Heidelberg (2005)
22. Watkins, C.: Learning from delayed rewards. PhD thesis, University of Cambridge (1989)

Seeing the Forest Through the Trees

Learning a Comprehensible Model from a First Order Ensemble

Anneleen Van Assche and Hendrik Blockeel

Computer Science Department, Katholieke Universiteit Leuven, Belgium
{anneleen.vanassche,hendrik.blockeel}@cs.kuleuven.be

Abstract. Ensemble methods are popular learning methods that are usually able to increase the predictive accuracy of a classifier. On the other hand, this comes at the cost of interpretability, and insight in the decision process of an ensemble is hard to obtain. This is a major reason why ensemble methods have not been extensively used in the setting of inductive logic programming. In this paper we aim to overcome this issue of comprehensibility by learning a single first order interpretable model that approximates the first order ensemble. The new model is obtained by exploiting the class distributions predicted by the ensemble. These are employed to compute heuristics for deciding which tests are to be used in the new model. As such we obtain a model that is able to give insight in the decision process of the ensemble, while being more accurate than the single model directly learned on the data.

Keywords: ensembles, first order decision trees, comprehensibility.

1 Introduction

In the process of knowledge discovery, one seeks to extract useful knowledge from data bases. But for knowledge to be useful, predictive accuracy is not sufficient: the extracted patterns also need to be understood by human users in order to trust them and accept them. Moreover users often construct models to gain insight in the problem domain rather than to obtain an accurate classifier only. For this reason, researchers have advocated for machine learning methods, such as decision tree learners and rule learners which yield comprehensible models. In the context of inductive logic programming, comprehensibility is usually even more important than in propositional learning because in the problem domains tackled by ILP algorithms (such as life science, medical domains etc.) end-user acceptance often depends on the learners ability to explain the reasoning behind its decisions.

For quite some years now, a lot of interest has been shown to a class of learning methods called ensembles. The main goal in the design of ensemble methods is to increase the predictive accuracy of the classifier and studies indeed have shown the discrimination power of ensemble methods both theoretically and empirically [1,5,8], and in propositional as well as relational learning [13,11,7,19]. Ensemble

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 269–279, 2008.

methods are learning algorithms that first construct a set of classification models and then classify new data points by combining the predictions of each of these models. Exactly by doing so, they are often able to increase the stability and predictive accuracy significantly over the single models. On the other hand the comprehensibility of the learned hypothesis drops significantly, since the result of an ensemble method is a large set of models with (weighted) votes connected to each of them, which obviously becomes very hard to interpret. This is one of the major reasons why ensemble methods are still less popular in ILP than in propositional learning.

Some authors have pointed out that striving for comprehensibility is one of the important issues in ensemble learning requiring future investigations [1,16]. Quite some successful research has been carried out already in this area. More in particular researchers have tried to obtain comprehensibility by means of extracting a new interpretable model from an existing ensemble without sacrificing too much accuracy. Craven and Shavlik [6] presented an algorithm TREPAN for extracting comprehensible, symbolic representations from trained neural networks. TREPAN extracts a decision tree using the network as an oracle to answer queries during the extraction process. Domingos [9] proposed Combined Multiple Models (CMM). CMM first builds an ensemble of multiple models and then reapplies the base learner to recover the partitioning implicit in the multiple model ensemble. This is achieved by giving the base learner a new training set, composed of a large number of examples generated and classified according to the ensemble. Zhou et al. [21] utilize neural network ensembles to generate new instances and then extract symbolic rules from those instances. Ferri et al. [10] describe a method to learn a comprehensible model from an ensemble by selecting the single hypothesis from a multi-tree that is most similar to the combined hypothesis according to a similarity measure. In the context of relation learning, Van Assche et al. [18] proposed a method similar to Domingos' to learn a new interpretable model from artificially generated relational data based on a first order ensemble.

The approaches described above all rely on artificially generated data to tackle the problem of finding an interpretable model that approximates the ensemble: either by classifying this new data by the ensemble and constructing a new interpretable model on it, or by using this new data to measure similarity between the ensemble model and candidate interpretable models. As was described in Van Assche et al. [18] generating relational artificial data is not straightforward (as opposed to propositional data), because data distributions are far more complex and examples do not longer have a fixed set of attributes. In Van Assche et al. [18] new 'partial' examples are constructed by adding more and more constraints (that occur as tests in a tree of the ensemble) to the example. Before a constraint is added, satisfiability needs to be checked with respect to already added constraints. Therefore, one needs to make use of the inherent equivalence relation between a defined concept and its definition. The algorithm relies on the users ability to adequately define this background knowledge in both directions

of the equivalence relation. But in Prolog, this becomes almost unfeasible when introducing aggregate or other complex functions.

For this reason, in this paper we aim to learn a single interpretable model from a first order ensemble without the need of generating artificial data nor requiring extra input from the user. Instead of first generating artificial data and computing class distributions for different possible tests on this data, class distributions are estimated directly from the information available from the different models in the ensemble in order to decide which tests are to be used in the new model. We describe the proposed approach in detail in the next section. Afterwards it is evaluated on some relational data sets comparing the accuracy and model size to the original single tree and the ensemble classifier. In the last section we formulate conclusions and some ideas for future research.

2 Proposed Method

In this section we propose a method to learn a single first order decision tree from a first order decision tree ensemble[1].

2.1 Computing Heuristics from the Ensemble

Assume E is an ensemble of N first order decision trees, which we would like to represent by one single first order decision tree. Actually a decision tree is able to represent any function described over the instance space as it can separate the instance space completely if necessary, so there also exist a decision tree that exactly represents the function described by the ensemble (but it cannot necessarily be learned from the training data). The decision boundaries of an ensemble of decision trees correspond to the union of decision boundaries of the decision trees it consists of and the label of each bounded area in the ensemble's decision space is a combination of the labels of the corresponding overlapping areas of each of the decision trees. As such, for a tree to represent this ensemble it suffices to use the tests that correspond to the decision boundaries in the decision space of the ensemble, meaning that it needs only tests that appear in the trees of the ensemble to represent the ensemble.

To construct a first order decision tree from an ensemble we will closely follow the procedure of regular first order tree induction according to TILDE [3] as shown in Table 1, though now certain subroutines will compute information from the ensemble instead of from the data (subroutines that will change are shown in boxes).

In TILDE, first order decision trees are learned with a divide and conquer algorithm similar to C4.5 [15]. The OPTIMAL_SPLIT procedure returns a query Q_b, which is selected from a set of candidates generated by the refinement operator ρ, by using a heuristic, such as information gain or gain ratio for classification problems, or variance reduction for regression.

[1] Though the more general framework does not only apply to first order ensembles, in [20] we evaluate it in a propositional context.

Table 1. TILDE algorithm for first order logical decision tree induction [3]

```
procedure GROW_TREE (E: examples, Q: query):
    candidates := ρ(← Q)
    ← Q_b := OPTIMAL_SPLIT(candidates, E)
    if STOP_CRIT (←Q_b, E)
    then
        K := PREDICT(E)
        return leaf(K)
    else
        conj := Q_b − Q
        E_1 := {e ∈ E|←Q_b succeeds in e ∧ B}
        E_2 := {e ∈ E|←Q_b fails in e ∧ B}
        left := GROW_TREE (E_1, Q_b)
        right := GROW_TREE (E_2, Q)
        return node(conj, left, right)
```

We will tackle the points where our algorithm differs from the normal top-down induction procedure in the next sections.

2.2 Generation of Candidate Test Queries

In the normal first order decision tree induction algorithm described in Table 1, candidate tests are generated by extending the current query Q (the conjunction of all succeeding tests from the root to the leaf that is to be extended) with a number of new literals that are specified in the language bias, using a refinement operator ρ which operates under θ-subsumption. A given node n in a first order tree may introduce variables that can be reused in the nodes of its left subtree, which contains the examples for which the conjunction in n succeeds (with certain bindings for these variables).

In order to represent the hypothesis of the ensemble by a single tree, on the other hand, it is sufficient to use the tests that were used in the ensemble, as pointed out in the previous section. In a first order decision tree, the tests in the internal nodes of the tree are (conjunctions of) first order literals, which might be bound to literals occurring higher in the tree. So to select the potential tests to construct the new tree, for each node in a tree of the forest, the conjunction of succeeding first order literals occurring from the root until that node is considered. Then this conjunction is split into separate tests by taking literals together that share variables. As such we get a fixed set of tests which can be used to put in the internal nodes of the new tree. This is in contrast with usual ILP hypothesis construction where the search space grows as new literals (that may introduce new variables) are added to the hypothesis. By using this candidate selection step which provides a fixed set of first order tests, we can efficiently

construct a new tree from the ensemble. As tests can consist of more than one literal at a time (it might be a chain of literals which share variables and that occurred on a path of a tree in the first order ensemble) actually some kind of lookahead is now allowed in the tree as conjunctions of literals might be added at once in a node. Moreover, this allows to represent the same concept in a more compact way than is the case when each node can only contain one literal at a time.

2.3 Computing the Optimal Split

In order to construct a tree that models the ensemble, in each node of the tree the optimal split needs to be chosen based on the ensemble instead of the data. Assume the ensemble E gives labels $L_E(x)$ to new examples x according to a combination of the predictions of each of its N base trees DT_k, as follows:

$$L_E(x) = argmax_{C_i}(\frac{1}{N}\sum_{k}^{N} P_{DT_k}(C_i|x)) \tag{1}$$

So it predicts the class C_i with the highest average predicted probability over the different N trees. Now we need to adapt the OPTIMAL_SPLIT procedure such that the heuristic used (assume information gain but gain ratio or other heuristics could be used as well) can be computed from the distributions in the ensemble.

Suppose Q is the conjunction of tests that occurred along the path from the root until node n, then the regular formula of information gain IG for a certain test t in n is

$$\begin{aligned} IG(t|Q) = entropy(Q) \quad & -P(t|Q)entropy(t,Q) \\ & -P(\neg t|Q)entropy(\neg t,Q) \end{aligned} \tag{2}$$

where

$$entropy(A) = \sum_{i=1}^{c} -P(C_i|A)\log_2 P(C_i|A) \tag{3}$$

with c the total number of classes and C_i the ith class and A any set of conditions.

While the usual top-down induction algorithm will estimate the class probabilities ($P(C_i|A)$) from the training data, we will now estimate them from the ensemble. A decision tree is constructed to model class distributions in the data and thus can be used to estimate these $P(C_i|A)$. Suppose we have a decision tree DT_k in the ensemble E, we can now estimate this $P_{DT_k}(C_i|A)$ by propagating it through the tree DT_k, applying the law of total probability in each node, until we end up in the leaves. Then we get:

$$P_{DT_k}(C_i|A) = \sum_{\text{leaves } l_{kj} \text{ in } DT_k} P_{DT_k}(C_i|Conj_{l_{kj}}, A)P_{DT_k}(Conj_{l_{kj}}|A) \tag{4}$$

where $Conj_{l_{kj}}$ is the conjunction of tests from the root of tree DT_k until leaf l_{kj}. The class probability estimate $P_E(C_i|A)$ of the ensemble E is then the average over the class probability estimates $P_{DT_k}(C_i|A)$ of the N trees in E.

In the equation 4 there is one term for each leaf in the tree. Now the probability $P_{DT_k}(C_i|Conj_{l_{kj}}, A)$ corresponds to the probability estimate $P_{DT_k}(C_i|Conj_{l_{kj}})$ given by leaf l_{kj} (by using the example frequencies in that leaf) because decision tree DT_k assumes that, as no tests were found to split the leaf l_{kj} further, the class C_i is conditionally independent from the tests in A given the tests in $Conj_{l_{kj}}$.

For the other probability $P_{DT_k}(Conj_{l_{kj}}|A)$, we can distinguish 3 possible cases: (assume B is the background knowledge defined for the problem domain)

- $A \wedge B \models Conj_{l_{kj}}$: then $P_{DT_k}(Conj_{l_{kj}}|A) = 1$ and $P_{DT_k}(C_i|A) = P_{DT_k}(C_i|Conj_{l_{kj}})$ (this means all examples succeeding A succeed $Conj_{l_{kj}}$ and would end up in leaf l_{kj})
- $A \wedge B \models \neg Conj_{l_{kj}}$: $P_{DT_k}(Conj_{l_{kj}}|A) = 0$ and leaf l_{kj} of tree DT_k will not contribute in the probability $P_{DT_k}(C_i|A)$ (this means all examples succeeding A will fail $Conj_{l_{kj}}$ and will not end up in leaf l_{kj})
- $A \wedge B \not\models Conj_{l_{kj}}, A \wedge B \not\models \neg Conj_{l_{kj}}$: $0 < P_{DT_k}(Conj_{l_{kj}}|A) < 1$ and leaf l_{kj} of tree DT_k partly contributes to the probability $P_{DT_k}(C_i|A)$ (this means part of the examples succeeding A might succeed $Conj_{l_{kj}}$ and end up in leaf l_{kj} (with probability $P_k(Conj_{l_{kj}}|A)$) and a part might not succeed $Conj_{l_{kj}}$ and end up in one of the other leaves)

To be able to estimate these probabilities $P_{DT_k}(Conj_{l_{kj}}|A)$, we could either make the assumption that the tests in $Conj_{l_{kj}}$ are conditionally independent from those in A (that are not given by $Conj_{l_{kj}}$), or compute them on a data set. The assumption in the first option is exactly the same as made by Quinlan [14], when classifying instances with missing values by a tree. This option then boils down to $P_{DT_k}(Conj_{l_{kj}}|A) = P_{DT_k}(Conj_{l_{kj}}|\{t_e : (A \wedge B \models t_e) \wedge (Conj_{l_{kj}} \wedge B \models t_e \vee Conj_{l_{kj}} \wedge B \models \neg t_e)\})$ assuming independence with tests t_e that do not fulfill $(A \wedge B \models t_e) \wedge (Conj_{l_{kj}} \wedge B \models t_e \vee Conj_{l_{kj}} \wedge B \models \neg t_e)$. This means that to find $P_{DT_k}(C_i|A)$ from equation 4 satisfiability needs to be checked between A and $Conj_{l_{kj}}$ for each of the leaves l_{kj}. As this is quite expensive to compute so many times, we gave preference to the other method estimating the probabilities from the available data[2]. We proceeded as follows: to find the probability that examples satisfying a certain test end up in a certain leaf l_{kj}, we check the proportion of examples, covered[3] by the tests $Conj_{l_{kj}}$, that is also covered by the test A. In the implementation, this is done by keeping track of precomputed coverlists for all leaves and candidate tests. These coverlists store for all available examples whether the examples are covered by the corresponding test or not. Deciding which tests end up in which leaves is then simply a matter of taking

[2] This data is either the training data or both training and unlabeled test data.
[3] An example is covered by a test if the test succeeds for that example.

the intersection of their coverlists, and as such this avoids expensive satisfiability testing between queries.

2.4 Stop Criteria

Using the method described above to compute the information gain for tests according to an ensemble E, a decision tree is built approximating the ensemble E, each time replacing a leave n, with $Conj_n$ the conjunction of tests occurring on the path from the root to leaf n, in the tree by an internal node as long as we can find a test t where $IG_E(t|Conj_n) \geq IG_E(t_i|Conj_n)$ for all possible tests t_i and $IG_E(T|Conj_n) > 0$. As conditional probabilities between tests are computed from the available data, the information gain of all tests will be zero if only one example of the available data ends up in the current node and no further splitting will be performed.

Nevertheless, at the end of the tree construction, some redundant splits will still be present in the tree, as they might change the class distributions in the leaves but not the eventual classes predicted by the leaves. To avoid this, tree construction will be stopped when all examples that end up in the current node are labeled the same by the ensemble.

2.5 Prediction of a Leaf in the New Tree

When no good tests are found to split the current node, or when a stop criterion is met, the node is turned into a leaf. The label predicted by such a leaf n with a conjunction of tests $Conj_n$ from the root to that leaf, is the label predicted by the ensemble according to the tests $Conj_n$, so:

$$L_n = argmax_{C_i}(\frac{1}{N}\sum_{k}^{N} P_{DT_k}(C_i|Conj_n)) \tag{5}$$

where the probabilities P_{DT_k} of the N trees of the ensemble are computed according to equation 4.

3 Empirical Evaluation

The method described above was implemented in the ACE-hipP system [4] (and named RISM, Relational Interpretable Single Model). The ACE-hipP system contains a first order decision tree learner TILDE [3], and some ensemble methods such as bagging and first order random forests [19] which use TILDE as the base learner.

We performed experiments on a Trains data set [12] generated with the Random Train Generator from Muggleton[4] according to a concept specified in Van Assche et al. [19], on the Carcinogenesis data set [17] and on the Financial data

[4] The train generator is available at http://www-users-csyork.ac.uk/~stephen/progol.html.

Table 2. Accuracy (in percentage) and model size on three data sets for Bagging and FORF(0.25) with 3-11-33-50 trees, the RISM model that was learned from these ensembles and TILDE

	Accuracy				Model size			
	3	11	33	50	3	11	33	50
				Trains data				
Bagging	67.5	70.7	71.5	71.5	136.9	511.8	1529.2	2319.1
RISM(Bagging)	67.0	68.0	68.0	68.6	47.7	47.3	45.4	42.5
RISM_u(Bagging)	68.3	70.1	70.4	70.4	77.8	85.5	82.2	81.1
FORF(0.25)	67.3	70.8	71.8	72.1	128.8	471.4	1416.4	2151.5
RISM(FORF(0.25))	65.5	68.8	70.4	70.1	48.8	42.5	39.8	39.6
RISM_u(FORF(0.25))	65.4	70.5	70.7	70.8	84.3	76.8	71.4	69.4
TILDE			68.9				40.6	
				Carcinogenesis data				
Bagging	59.3	62.2	62.0	62.5	106.7	390.1	1176.6	1790.6
RISM(Bagging)	58.5	59.7	61.3	61.1	52	47.8	49	49.1
RISM_u(Bagging)	60.3	61.9	63.1	63.2	80.1	79.3	80.5	67.1
FORF(0.25)	60.4	61.9	62.0	62.6	104.3	380.2	1138	1711.6
RISM(FORF(0.25))	59.7	61.4	60.9	62.1	50.7	47.3	46.7	44.6
RISM_u(FORF(0.25))	60.1	62.1	62.3	63.1	80.6	77.8	73.8	58.6
TILDE			61.1				29.9	
				Financial data				
Bagging	84.3	86.7	86.6	86.2	35.9	131	397.1	600.3
RISM(Bagging)	83.2	84.7	84	83.3	14.5	12	10.4	10.6
RISM_u(Bagging)	84.4	86.5	86.3	86.1	26.3	24.4	21.3	20.5
FORF(0.25)	84.6	86.5	86.5	86.3	41.1	151.8	449.5	688.8
RISM(FORF(0.25))	84.6	85	85.1	84.4	12.3	10.4	9.3	7.3
RISM_u(FORF(0.25))	85.1	86.1	86.1	86.5	23.8	20.6	18.9	17.7
TILDE			84.5				10.5	

set [2]. We constructed a bagged ensemble with 3, 11, 33 and 50 trees, as well as a random forest where 25% of the features were used at each node (FORF(0.25)). From these ensembles an interpretable single model was constructed applying the method described above (either using only the training set (RISM) or using both the training and unlabeled test set (RISM_u) to compute probability estimates). For comparison, we also built a single (pruned) tree directly on the training data. As the heuristic implemented for RISM is information gain, also information gain was used to build the trees in the ensembles and the single tree directly on the data. All experiments were done averaging over 10 different 5-fold cross-validations.

In table 2 we report test accuracy (in percentage) and model size (in terms of number of nodes in the trees) for the different settings. We provide results for different ensemble sizes to show the evolution in accuracy of the ensembles and RISM when more trees are added to the ensemble. For comparison between the

different algorithms on the other hand, we focus on the results for ensembles of 33 (or 50) trees, because by then, accuracy should somehow be leveled out.

As can be seen from the table, the accuracy results of RISM(_u) increase when more trees are added as is also the case for the ensemble methods Bagging and FORF. More surprisingly, while the model size of the ensembles obviously increases when more trees are added, the model size of RISM(_u) almost always decreases. So when adding more trees, both the accuracy of the obtained single model increases and the size decreases. This is because the probability estimates derived from the ensemble to compute heuristics for the single tree improve when the ensemble comprises more trees. When only the training set is used to compute probability estimates from the ensembles we find that overall RISM(FORF(0.25)) slightly outperforms RISM(Bagging) while FORF(0.25) does not necessarily outperform Bagging. Also the models obtained by RISM(FORF(0.25)) are slightly smaller than those of RISM(Bagging). This seems to indicate that the increase in diversity among the trees of FORF over Bagging improves the probability estimates. While RISM(FORF(0.25)) is only able to improve slightly over TILDE (and not even always), when unlabeled test instances are also used to estimate probabilities from the ensembles, RISM_u obtains accuracies comparable to the ensembles it is deduced from. Sizes of models constructed by RISM_u are more or less double the size of those of RISM because more examples are sorted down the trees as unlabeled test instances are now used as well. This means that on average in each branch one extra node is added.

4 Conclusions and Future Work

In this paper we presented a method to learn a first order decision tree that approximates the decisions made by an ensemble of first order decision trees. The tree is obtained without the need of generating artificial data nor requiring extra input from the user. Instead, first a fixed set of possible candidate tests for the nodes in the new tree are extracted from the ensemble. Next, heuristics are computed for each of the candidate tests by predicting class distributions for these tests using the ensemble. Labels in the eventual leaves of the new tree are the labels predicted by the ensemble. As such, we aim to obtain an interpretable tree that is able to give insight in the predictions of the ensemble, while being more accurate than a single tree directly learned on the data. Experiments show that when only training data is used to compute conditional probabilities from the ensemble, improvements over building a single tree directly on the data are minimal. On the other hand, when unlabeled test instances are used as well, the single model deduced from the ensemble obtains accuracies comparable to the ensemble's, while on average only having one extra node per branch compared to a tree directly built on the data.

As currently no postpruning is performed on the obtained trees, it would still be interesting to see what the effect is on the accuracy of applying further postpruning and/or stop criteria. Furthermore, we would also like to investigate the influence on the stability of the trees.

Acknowledgements

Anneleen Van Assche is supported by the Institute for the Promotion of Innovation by Science and Technology in Flanders (I.W.T.-Vlaanderen). Hendrik Blockeel is Postdoctoral Fellow of the Fund for Scientific Research - Flanders (Belgium) (F.W.O.-Vlaanderen).

References

1. Bauer, E., Kohavi, R.: An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. Machine Learning 36, 105 (1999)
2. Berka, P.: Guide to the financial data set. In: Siebes, A., Berka, P. (eds.) The ECML/PKDD 2000 Discovery Challenge (2000)
3. Blockeel, H., De Raedt, L.: Top-down induction of first order logical decision trees. Artificial Intelligence 101(1-2), 285–297 (1998)
4. 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. Journal of Artificial Intelligence Research 16, 135–166 (2002)
5. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
6. Craven, M.W.: Extracting Comprehensible Models from Trained Neural Networks. PhD thesis, University of Wisconsin, Madison (2003)
7. de Castro Dutra, I., Page, D., Costa, V., Shavlik, J.: An empirical evalutation of bagging in inductive logic programming. In: Matwin, S., Sammut, C. (eds.) ILP 2002. LNCS (LNAI), vol. 2583, pp. 48–65. Springer, Heidelberg (2003)
8. Dietterich, T.: Ensemble methods in machine learning. In: Kittler, J., Roli, F. (eds.) MCS 2000. LNCS, vol. 1857, pp. 1–15. Springer, Heidelberg (2000)
9. Domingos, P.: Knowledge discovery via multiple models. Intelligent Data Analysis 2, 187–202 (1998)
10. Ferri, C., Hernández-Orallo, J., Ramírez-Quintana, M.: From Ensemble Methods to Comprehensible Models. In: Lange, S., Satoh, K., Smith, C.H. (eds.) DS 2002. LNCS, vol. 2534, pp. 165–177. Springer, Heidelberg (2002)
11. Hoche, S., Wrobel, S.: Relational learning using constrained confidence-rated boosting. In: Rouveirol, C., Sebag, M. (eds.) ILP 2001. LNCS (LNAI), vol. 2157, pp. 51–64. Springer, Heidelberg (2001)
12. Michalski, R.: Pattern Recognition as Rule-Guided Inductive Inference. IEEE Transactions on Pattern Analysis and Machine Intelligence 2, 349–361 (1980)
13. Quinlan, J.: Boosting first-order learning. In: Arikawa, S., Sharma, A.K. (eds.) ALT 1996. LNCS, vol. 1160, Springer, Heidelberg (1996)
14. Quinlan, J.R.: Induction of decision trees. Machine Learning 1, 81–106 (1986)
15. Quinlan, J.R.: C4.5: Programs for Machine Learning. In: Machine Learning. Morgan Kaufmann series, Morgan Kaufmann, San Francisco (1993)
16. Ridgeway, G., Madigan, D., Richardson, J., adn O'Kane, T.: Interpretable boosted naive bayes classification. In: Proc. of the 4th International Conference on Knowledge Discovery in Databases, pp. 101–104. AAAI Press, Menlo Park (1998)
17. Srinivasan, A., King, R., Muggleton, S., Sternberg, M.: Carcinogenesis predictions using ILP. In: Lavrač, N., Džeroski, S. (eds.) ILP 1997. LNCS (LNAI), vol. 1297, pp. 273–287. Springer, Heidelberg (1997)
18. Van Assche, A., Ramon, J., Blockeel, H.: Learning interpretable models from an ensemble in ILP. In: Proc. of the 16th International Conference on Inductive Logic Programming – short papers, pp. 210–212 (2006)

19. Van Assche, A., Vens, C., Blockeel, H., Džeroski, S.: First order random forests: Learning relational classifiers with complex aggregates. Machine Learning 64(1-3), 149–182 (2006)
20. Van Assche, A., Blockeel, H.: Seeing the forest through the trees: Learning an interpretable model from an ensemble. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) ECML 2007. LNCS (LNAI), vol. 4701, Springer, Heidelberg (2007)
21. Zhou, Z., Jiang, Y., Chen, S.: Extracting symbolic rules from trained neural network ensembles. AI Communications 16(1), 3–15 (2003)

Building Relational World Models for Reinforcement Learning

Trevor Walker[1], Lisa Torrey[1], Jude Shavlik[1], and Richard Maclin[2]

[1] University of Wisconsin, Madison WI 53706, USA
[2] University of Minnesota, Duluth, MN 55812, USA

Abstract. Many reinforcement learning domains are highly relational. While traditional temporal-difference methods can be applied to these domains, they are limited in their capacity to exploit the relational nature of the domain. Our algorithm, AMBIL, constructs relational world models in the form of relational Markov decision processes (MDPs). AMBIL works backwards from collections of high-reward states, utilizing inductive logic programming to learn their *pre-image*, logical definitions of the region of state space that leads to the high-reward states via some action. These learned preimages are chained together to form an MDP that abstractly represents the domain. AMBIL estimates the reward and transition probabilities of this MDP from past experience. Since our MDPs are small, AMBIL uses value-iteration to quickly estimate the Q-values of each action in the induced states and determine a policy. AMBIL is able to employ complex background knowledge and supports relational representations. Empirical evaluation on both synthetic domains and a sub-task of the RoboCup soccer domain shows significant performance gains compared to standard Q-learning.

1 Introduction

We present a relational reinforcement learning (RL) method, AMBIL (Abstract Model Building via ILP), that can handle complex relational reinforcement learning domains with real-valued, high-dimensional feature spaces and sparse reward structures. Via its novel method for partitioning an environment into useful states, AMBIL attempts to find good policies by building a model of a domain's state-transition structure in the form of an abstract Markov decision process (MDP). AMBIL uses inductive logic programming (ILP) extensively to learn the states in this MDP, treating each state as a separate ILP learning problem. The use of ILP techniques provides power and generality to our models that would otherwise be unattainable with other approaches.

Traditional RL methods, such as model-free, temporal-difference (TD) learning algorithms [15], find optimal or near-optimal policies to maximize the reward received while acting within the domain. However, as the domain becomes more complex, these methods often rely upon function approximation for effective generalization of training data.

In complex domains, generalization by function approximators can pose significant difficulties and typically requires a large number of examples. AMBIL attempts to

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 280 – 291, 2008.

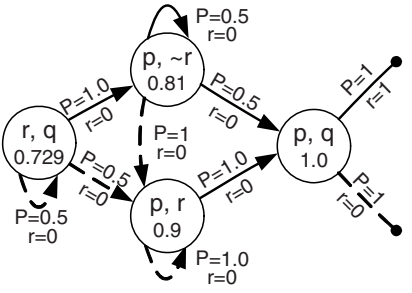


Fig. 1. Sample MDP produced by AMBIL. First-order logical rules define each state. Arcs represent action transitions for two actions (one action with solid and one with dotted lines). Arcs show the probability of transitions, given the action, and immediate rewards. Value-iteration, with γ=0.9, calculates the Q-value of each state-action pair (not shown). The maximum Q-value from each state (shown inside each state) determines the policy for the state.

improve generalization by focusing on the state transitions seen in the training data. AMBIL uses ILP to partition the state space and, since these partitions are based directly upon the state transitions of actions, can generalize the examples more effectively. AMBIL also exploits the relational aspects of RL domains; similar domain actions can be abstracted and learned together rather than one at a time. AMBIL abstracts objects in the domain, learning rules that apply to whole classes of objects.

The partitions learned by AMBIL form a relational MDP, with the transition and reward function estimated from previously observed data. Existing techniques, such as value-iteration [15], provide estimates of the expected reward for each MDP state. AMBIL's MDP both determines the policy and provides an explicit, easy-to-analyze representation of the domain.

AMBIL is not the first method to address relational RL. Several techniques exist that apply relational function approximation to traditional Q-learning or dispense with function approximation altogether. We compare AMBIL to these methods in the related-work section.

We present empirical results in a synthetic domain and in the challenging Breakaway subtask of the RoboCup soccer domain [8], demonstrating faster learning and higher asymptotes at convergence than traditional Q-learning reinforcement learning algorithms.

2 Background

In reinforcement learning [15], an agent performs actions in some domain. After each action, the agent receives feedback in the form of numeric rewards. The agent attempts to learn a policy π that maps domain states to actions to maximize the sum of rewards received.

A reinforcement learning domain can be thought of as a Markov Decision Process (MDP). An MDP is a five-tuple $\langle S,A,P,R,\gamma \rangle$ where S is a finite set of states; A is a finite set of actions; P is a transition function denoting the next-state distribution after taking action a in state s; R is a bounded reward function denoting the expected immediate reward received for taking action a in state s; and $\gamma \in [0,1)$ is a discount factor.

In many cases, the underlying MDP for a domain is unknown. In these situations, the agent must interact with the domain in order to learn a policy. Either the agent can learn a policy directly, without learning the underlying MDP, or the agent can attempt to learn the underlying structure in some form and then create a policy based upon the learned structure. The former approach is called model-free or direct reinforcement learning and the latter is called model-learning.

Q-learning [16], a common model-free approach used in RL when the underlying MDP is not known, makes few assumptions or restrictions concerning the RL domain and performs competitively with many model-learning approaches. It works by learning a function $Q(s,a)$ that represents the expected reward for taking action a in state s. Q-learning determines policy by choosing the action with the highest Q-value for a given state. The Q-function itself is often modeled using a function approximator, allowing Q-learning some ability to scale to complex domains with real-valued, high-dimensional feature spaces.

3 Building World Models

AMBIL, a model-building technique, partitions the state space and creates an MDP from the partitions. First-order logical rules define the portion of state space each

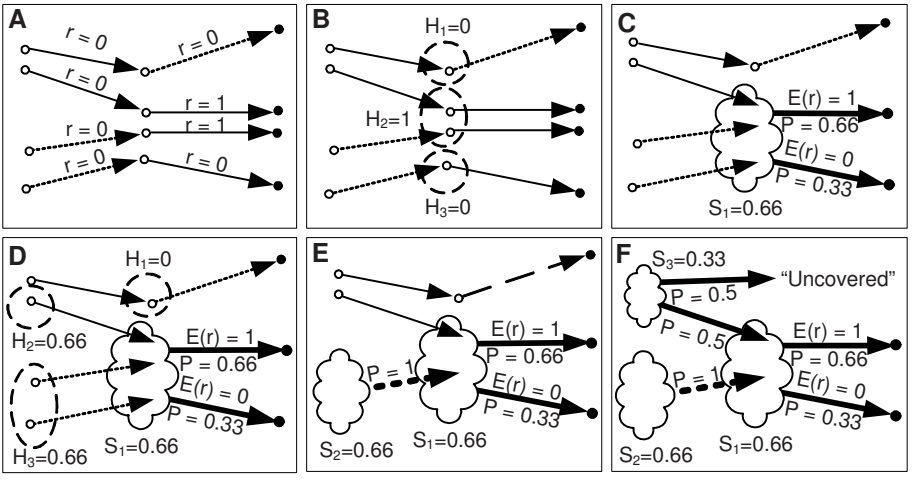


Fig 2. Sample MDP being built in a 2D continuous state space. A. Example states (open dots), reached by two actions (solid and dotted arcs). Actions reaching filled dots terminated the episode. B. Initial terminating preimages considered for learning, with their heuristic scores H_i shown, based upon the rewards in the example data. C. An MDP state learned based upon preimage H_2. Hote the generalized state S_1 could cover more example states than intended. Action arcs show the aggregate reward and transition functions for S_1. All calculations use γ=1. D. Next stage of preimage selection and scoring. E. MDP extended by generalizing preimage H_3 into MDP state S_2. F. Final MDP after all preimages have been generalized. Note some transitions, such as the top one from S_3, may not lead to a learned state and are placed in a special "uncovered" state, with a score of zero.

partition covers. The MDP in turn leads to a policy intended to maximize the rewards received. AMBIL then exploits this policy to collect more examples from the domain. AMBIL is a batch algorithm. It iterates between building models and utilizing those models in the domain. Figure 1 depicts a sample MDP produced by AMBIL. The following sections explain the model-building process.

3.1 Terminology

AMBIL operates on a collection of examples containing sequences of {*state, action, reward*} tuples from the RL domain we are attempting to solve. We will refer to these as *example states*. Additionally, as we partition the example state space, the partitions become states in an MDP. We will refer to these states as *MDP states*.

AMBIL bases its partitioning upon *preimages* of the current, partially-built MDP. The preimage of an MDP state *s* for action *a* is the set of example states *s'* that lead to *s* via action *a*. A *terminating preimage* is the set of example states *s'* in which action *a* was taken, resulting in the termination of an episode with immediate reward of $r \pm \delta$. For example, the terminating preimage of shots scored would be all example states in which the agent shot, resulting in a score and an immediate reward of +1, ending the episode.

3.2 Algorithm Overview

Table 1 presents the AMBIL algorithm and Figure 2 depicts several stages of the algorithm in operation.

When partitioning example space, AMBIL begins with an empty MDP model. It then examines all terminating preimages and greedily selects one that scores the highest according to a heuristic (see Figure 2B). Once a preimage is selected, AMBIL uses ILP to generalize the example states in the preimage into one or more first-order logical rules, which may be include some negative examples in domains that are stochastic. Each rule becomes a single MDP state, covering all of the example states that match its logical rule. Each time AMBIL adds an MDP state, it calculates the transition and reward functions for that state by examining the example states covered by the rule (see Figure 2C). Then, AMBIL applies value-iteration to obtain the score for each state currently in the MDP. This process repeats, greedily selecting from all previously unlearned terminating preimages and MDP state preimages (see Figure 2D-F), until the MDP covers at least a user-specified fraction of the example states.

Table 1. AMBIL's MDP Model Building Algorithm

Input: *E*: set of example states
BK: background knowledge expressed in first-order predicate calculus

Do:

1. Initialize MDP to empty model
2. Score possible preimages and greedily select best one to learn
3. Learn rule(s) via inductive logic programming based upon selected preimage, *E*, and *BK*
4. Add learned rule(s) to MDP as new MDP state
5. Estimate MDP's reward functions and transition probabilities
6. Calculate *Q*-values for all states in MDP
7. If examples states sufficiently covered, stop. Else go to step 2

Return: MDP

3.3 Preimage Selection

When constructing an MDP, AMBIL greedily selects preimages to define the MDP. These preimages consist of example states not currently covered by some MDP state and lead to a previously learned MDP state or a terminating preimage.

A simple heuristic scores each eligible preimage by an *optimistic* Q-value using Bellman-backups [1], according to Equation 1. The optimistic Q-value represents the expected Q-value of the preimage's constituent example states. Thus, the value is the expected reward of a transition from an example state in the preimage to the destination state of the MDP via action a. This value is optimistic since it assumes AMBIL can learn the preimage exactly and that the example data is an i.i.d. sample of all possible example states in the preimage. In RL domains, this will not be true, since the distribution of example states visited depends upon the policy of the learner. In addition, the generalized rule(s) learned to represent the preimage will often cover parts of example space that were not part of the preimage. Even though the optimistic Q-value is inaccurate, it serves as a good heuristic to guide preimage selection.

$$Q_{opt}(\text{Preimage}(\text{S}, a)) = \frac{\sum_{s \in \text{Preimage}(\text{S},a)} r(s,a) + \gamma Q(\text{S})}{\left|\text{Preimage}(\text{S}, a)\right|} \quad (1)$$

Given the current MDP, only a subset of the possible preimages are eligible for learning. AMBIL ignores preimages learned previously. Each preimage must contain a minimum number of example states, guaranteeing that enough data will be available for the rule-learning stage. Preimages must also obtain a minimum optimistic Q-value score. This eliminates preimages unlikely to result in an increase in performance.

AMBIL exploits the relational nature of the domain during preimage selection. It considers preimages with multiple actions whenever the user indicates that two or more actions share similar behavior. In these cases, the preimage will be parameterized appropriately for each action. For example, two shooting actions, such as shoot(left) and shoot(right) used in Section 4's experiments, might be considered together as shoot(GoalPart), where GoalPart is a variable parameterizing the portion of the goal the shot was aimed at. This grouping allows AMBIL to exploit the relational nature of some actions. Even when these groupings exist, AMBIL also considers the single-action preimages since parameterized concepts may be more difficult to learn or specialized versions of actions might be needed.

When AMBIL first creates an MDP, the MDP will contain no states to use as a basis for preimages. Thus, AMBIL currently uses terminating preimages to initiate the MDP-building process. In the domains we have focused on, clearly defined terminating preimages exist (such as shots resulting in a scored goal). AMBIL could also use a domain's reward-structure information, if available, to determine the initial preimages. In non-sparse or infinite-horizon domains with no user-provided objective, AMBIL could cluster the sampled rewards to determine initial concepts.

3.4 Learning Concepts Via ILP

For each preimage selected, AMBIL uses inductive logic programming to generate first-order rules that describe the set of states that the preimage covers. Given an

```
shot_score(GoalPart) :-
   x_distance_wrt_kick_at_goal (GoalPart) > 6.0
   y_distance_wrt_kick_at_goal(GoalPart) > 2.0
   angle_between_kick_&_goalie(GoalPart) < 129.
```

Example 1. Learned rule for shot_scored preimage, with a parameter to represent both the shoot(left) and shoot(right) actions. The variable GoalPart allows this rule to be applied to shooting at either side of the goal, both during learning and problem solving.

example state, the learned rules classify whether or not it is in the preimage, i.e. whether it leads to the relevant MDP state via the indicated action or not. Although AMBIL could use simpler propositional methods to learn the preimage classification, ILP allows the relational aspect of the domain to be exploited: similar actions can be parameterized and learned as a single concept, similar domain objects can be generalized, and extensive background knowledge can be utilized to aid in describing the preimages. Currently, AMBIL uses the ILP system Aleph [12].

AMBIL selects the positive and negative examples based upon the preimage being learned. For a preimage(*s,a*), the example states that transitions to MDP state *s* via action *a*, AMBIL collects, as positive examples, all example states where the action *a* was taken and the following example state is covered by MDP state *s*. The negative examples are the example states where action *a* was taken and the following state is not covered by *s*. Example states in which the action *a* was not taken are ignored. There are generally many more negatives than positives. AMBIL uniformly subsamples the positive and negative examples to reduce Aleph's runtime. Typically, we subsample down to 500 total examples.

The positive and negative examples AMBIL provides to Aleph are in the form of sets of first-order predicates. As such, the examples can be parameterized to contain additional information. As mentioned in Section 3.3, this allows AMBIL to learn multiple actions as a single preimage. For example, the preimage "shots that score a goal" can be parameterized to include the shot destination as an argument, such as shot_scored(GoalPart). The user must specify which actions are similar and what parameters they require, but once that is done, AMBIL handles all of the parameterizations automatically.

Example 1 shows a parameterized rule learned for a soccer domain involving two shoot actions, shoot(left) and shoot(right). In this example, the positives contained all examples that shot left or right and scored. The negative set contained all examples that shot left or right and did not score.

3.5 Building the MDP

Given a learned preimage classification theory from ILP, AMBIL extends the MDP model by adding states, one for each rule in the theory. We treat the separate rules

from a theory independently since they may represent different aspects of the learned preimage and doing so increases the specificity of the final model.

While AMBIL treats the learned model as an MDP, the model might not actually be one. Often, the created model violates the Markovian assumptions required by MDPs and might be better characterized as a partially observable Markov decision process. However, in our experiments, even when the Markovian assumptions were clearly violated, treating the model as an MDP still yielded good results and made the calculation of the value function much faster.

In a proper MDP, states are discrete and disjoint. However, many of the rules generated by Aleph overlap with either other rules within a single preimage's theory or other states previously added to the MDP. In order to enforce disjointness among MDP states, AMBIL orders the states by their creation order, essentially creating an IF-ELSEIF-ELSE structure used to determine in which MDP state an example state belongs. When AMBIL adds multiple rules from a single preimage's theory, the created states are ordered according to their accuracy on the Aleph training set.

After adding states to the MDP, AMBIL calculates the MDP's state-transition probabilities and reward functions. AMBIL computes the transition probability from MDP state *s* to MDP state *s'* via action *a* by counting the number of example states that are covered by *s* and lead to *s'* via action *a*, normalizing these counts by the number of times action *a* was taken in state s. Similarly, it calculates the expected reward for action *a* from state *s* by averaging over the rewards seen in the training data. AMBIL employs *m*-estimates to condition the transition probabilities. We use m=5 in our experiments. In some cases, example states will exist that are not covered by any MDP state. AMBIL assigns these states to a special default state called the *uncovered* MDP state.

After it calculates the transition probabilities and rewards, AMBIL performs value-iteration [15] for all states in the MDP, except the special uncovered MDP state. The uncovered state's *Q*-value is some domain-dependent "background" score (e.g., zero). This discourages actions that would lead to the uncovered state. Although not necessary in our experiments, AMBIL could utilize refinements of the standard value-iteration algorithm, such as prioritized sweeping.

When adding states to the MDP, AMBIL must address several additional considerations. If two or more states overlap, states added later may not have adequate data. If the amount of data available to an MDP state is below some minimum threshold (currently, five example states), AMBIL discards that particular state.

On occasion, when AMBIL adds a state, the policy action it recommends (the argmax *Q* over all actions for this state) is not the same as the action of the original preimage. When the action does not match the preimage, this is indicative of one of two things: either a state does not have adequate data or Aleph improperly generalized the preimage. In these cases, AMBIL could discard the MDP state. However, in our experiments, this occurs infrequently and discarding the states is unnecessary.

3.6 The RL Learning Cycle

The previous sections described the process AMBIL uses to build a single MDP (and the associated policy). In the complete learning cycle for a given RL domain,

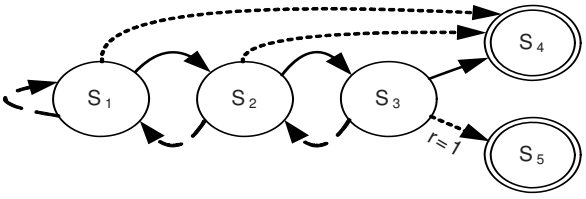


Fig. 3. Synthetic domain MDP. Arcs represent state-transitions for three separate actions. All actions are deterministic. S_4 and S_5 are terminating state. All rewards are zero, except for the single action leading from S_3 to S_5.

AMBIL first gathers some initial data by interacting with the domain. It then repeatedly generates an MDP with a corresponding policy and interacts with the domain following the new policy to gather more data. In each iteration, AMBIL attempts to generate a new, higher-scoring policy.

Since the AMBIL model-building process is computationally expensive, we may not want to rebuild a full model whenever new data is available. However, updating the reward and transition functions for the MDP between full builds is computationally feasible and does result in some improvement of the policy. We use this approach in the empirical results section below.

To gather initial data AMBIL either explores the domain randomly or uses another RL learning algorithm (e.g.) standard model-free Q-learning as a bootstrap. In domains with very sparse rewards under a random policy, the bootstrapping process is preferred since it is more likely to obtain informative data.

After an MDP exists, when interacting with the RL domain, given an example state, AMBIL determines the action to perform by matching the example state against the states in the MDP model. The argmax of the Q-values for all actions for a given state determines the policy for that state. Additionally, as done by standard RL algorithms, AMBIL performs a small fraction of exploratory actions.

4 Empirical Results

We present empirical results in two domains: a synthetic RL domain and the Breakaway RL domain [8], a subtask of the RoboCup soccer domain. We compare AMBIL with the standard SARSA(λ) [15] algorithm and with a model-learning approach based upon a Dyna-Q architecture [14]. We choose Dyna-Q as a control because it is an established approach for creating models of an environment in order to speed up RL.

4.1 Domains

The synthetic domain, shown in Figure 3, is a simple five-state, three-action, nondeterministic MDP, with two numeric features (drawn from overlapping uniform distributions), and a sparse reward structure, with the only reward occurring in one of two terminating states. The feature values for each state purposely overlap to simulate uncertainty in determining the underlying MDP state. We provided the learners

no direct knowledge of the underlying MDP. They must interact with the domain to obtain information.

The Breakaway domain is a 2-on-1 soccer end-game task based upon the RoboCup soccer simulator. In Breakaway, *M* attackers attempt to score on *N* defenders, including a goalie. Only the attacker with the ball makes policy decisions, while the attackers without the ball and the defenders follow hard-coded policies. The reward structure for this domain is very sparse, with a reward of one when a goal is scored and zero all other times. Each Breakaway episode is limited to 10 seconds, after which the episode ends with a zero reward. The Breakaway domain is highly non-deterministic with many real-valued features (for 2-on-1, there are 27 features).

Our Breakaway state representation is that of Maclin *et al.* [8]. For the *Q*-learner, we discretize these features into 32 overlapping intervals called *tiles*, each of which becomes a Boolean feature. Stone and Sutton [13] used this enhancement in RoboCup; tiling allows linear function approximators to represent non-linear concepts.

For Breakaway, the AMBIL background knowledge consists of *feature_less_than*, *feature_greater_than*, *feature_in_range*, and *feature_not_in_range* predicates. Additionally, the background includes predicates that provide information relative to passes and kicks. For example, *x_distance_wrt_kick* measures the distance from the kicker to an object along the direction of the kick. The background knowledge was designed to allow Aleph to discretize the base features, rather than add additional high-level domain knowledge.

4.2 Learning Algorithms

All three learning algorithms, AMBIL, SARSA(λ), and Dyna-Q, are implemented as batch learners and, as much as possible, we used the same tuning parameters on the standard *Q*-learner, giving the benefit of doubt to the experimental control. Parameters were tuned on the Q-learning base line and used for the other learners. Each learner "batch learns" every 25 games. All use an exploration rate of 1% and a discount factor of 0.97. For the *Q*-learner and Dyna-Q we used a λ setting, with λ=1 for recent example states decaying to λ=0 after a fixed number of games (200 for Breakaway, 50 for the synthetic MDP).

Both AMBIL and Dyna-Q use the standard Q-learning algorithm until enough data is available to start the respective algorithm. Fifty and 250 games are played prior to Dyna/AMBIL running for the synthetic and Breakaway domains, respectively.

Dyna-Q creates models that predict the feature values in the next state and the reward function of the domain directly from the data and then uses the models to train *Q*-functions. We modeled the reward function using a C4.5 decision tree [10]. Next state feature values are modeled independently of each other using support vector regression. We attempted to make these models as accurate as possible, although modeling high-dimensional, real-valued environments is known to be difficult. The next state models created by Dyna-Q are used to create synthetic examples. These examples are then utilized in the same manner as real examples. We created enough synthetic data to maintain a 4-1 ratio of actual examples to synthetic examples.

For the Breakaway domain, we also attempted to implement an RRL algorithm [5], a combination of traditional Q-learning with a relational TILDE-RT [2] function approximator. We were unable to obtain results better than random walks with this approach.

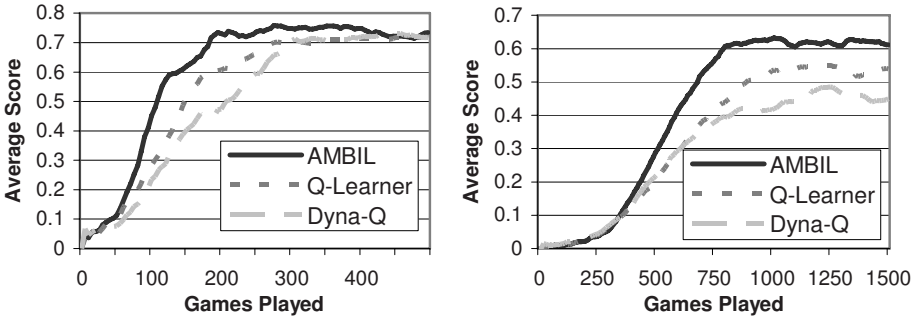


Fig. 4. Empirical Results. (Left) Synthetic Domain averaged reward received per game, averaged over previous 50 games. (Right) 2-on-1 Breakaway average reward received per game, averaged over previous 250 games.

4.3 Results

Figure 4 shows the average reward per game for the synthetic domain and for the 2-on-1 Breakaway domain. For both domains, we performed 10 runs of each algorithm and averaged the results. The reward per game is averaged over the previous 50 games for the synthetic domain and previous 250 games for the Breakaway domain.

In both domains, AMBIL outperformed both the *Q*-learner and the Dyna-Q algorithms, both in terms of early learning rate and asymptotic performance.

Several factors contribute to AMBIL's early performance gains over the Q-learner. The background knowledge we provide to AMBIL offers it an advantage. We attempted to provide propositionalized versions of the background knowledge to the standard Q-learner, but this resulted in worse performance, presumably due to overfitting allowed by the greatly increased number of features.

Beyond the background knowledge, AMBIL's models, by construction, focus on reaching high-reward states immediately and generalizing accurately. In Q-learning, on the other hand, reward information propagates slowly through the model by the means of SARSA(λ) backups and generalization performed by function approximation can be inaccurate, especially early in the learning curve.

The Dyna-Q implementation performed poorly in both domains. This was due to the difficulty of modeling the underlying domain directly. It is the difficulty of this type of modeling that motivated AMBIL's approach.

5 Related Work

Reinforcement learning using TD methods has been studied extensively. Sutton and Barto [15] provide an excellent summary of the basic techniques.

Dietterich and Flann [3] introduced the concept of using chains of preimages in their explanation-based reinforcement learning. Their action chaining approach shares

some basic similarities with AMBIL. However, their approach requires an accurate definition of the action consequences.

Kersting *et al.* [7] create abstract relational MDPs with many similarities to our own models. However, their approach to learning abstract MDPs requires the underlying MDP. We essentially provide a learning method capable of learning a similar abstract MDP in complex domains without knowledge of the underlying MDP.

Morales' [9] rQ-learning provides a state-abstraction approach to reinforcement learning, although their approach does not use an MDP representation. Unlike AMBIL, which learns the abstract states, Morales' approach requires user-provided abstractions. However, rQ-learning supports STRIPs-like operators with more richness then AMBIL's actions and provides a learning algorithm for refining these operators.

Van Otterlo's [11] CARCASS system provides a relational MDP representation, similar to AMBIL's, and provides methods to score and use the resulting MDP based upon interaction with the domain. Like rQ-learning, Van Otterlo's methods assume user-provided abstractions.

As an alternative approach to building an MDP, Džeroski *et al.* [5] proposed using a relational decision tree, such as TILDE [2], during Q-learning. Like AMBIL, this allows for both the integration of background knowledge and the exploitation of the relational aspects of actions and objects in the domain. However, like Q-learning, their approach represents only the long-term expected reward and does not model the immediate reward or transition information, while AMBIL does. Furthermore, they are still performing function approximation, which can be difficult for RL. Lecoeuche [6] and Driessens *et al.* [4], among others, further refined this approach.

Another recent approach to relational reinforcement learning, by Zettlemoyer *et al.* [17], also models the domain without building an MDP. Instead, they learn probabilistic STRIPs-like rules. A probabilistic planner uses these rules to solve the domain. Like AMBIL, they focus on the state transitions resulting from observed actions, although the rule learning process and final model does not resemble AMBIL's.

6 Conclusions and Future Work

Models of reinforcement learning domains allow faster learning than model-free Q-learning methods, as demonstrated in our empirical study, and provide information about the structure of the domain. Our algorithm, AMBIL, builds MDPs via inductive logic programming techniques, focusing on areas of high reward to guide search. The use of ILP techniques allows our models to represent relational domains, with abstraction of both objects and actions, and allows the incorporation of background knowledge.

Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful comments. This research is supported by DARPA IPTO under contract FA8650-06-C-7606.

References

1. Bellman, R.E.: Dynamic Programming. Princeton University Press, Princeton, New Jersey (1957)
2. Blockeel, H., De Raedt, L.: Top-down induction of first-order locical decision trees. Artificial Intelligence (June 1998)
3. Dietterich, T., Flann, N.: Explanation-based learning and reinforcement learning: A unified view. In: Proceedings of the International Conference on Machine Learning (1995)
4. Driessens, K., Ramon, J., Blockeel, H.: Speeding up relational reinforcement learning through the use of an incremental first order decision tree algorithm. In: Proceedings of the European Conference on Machine Learning (2001)
5. Džeroski, S., De Raedt, L., Blockeel, H.: Relational reinforcement learning. In: Proceedings of the International Conference on Machine Learning (1998)
6. Lecoeuche, R.: Learning optimal dialog management rules by using reinforcement learning and inductive logic programming. In: Proceedings of the North American Chapter of the Association of Computational Linquistic (June 2001)
7. Kersting, K., Van Otterlo, M., De Raedt, L.: Bellman goes relational. In: Proceedings of the International Conference on Machine Learning (2004)
8. Maclin, R., Shavlik, J., Torrey, L., Walker, T., Wild, E.: Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In: Proceedings of the Twentieth Conference on Artificial Intelligence (2005)
9. Morales, E.F.: Scaling up reinforcement learning with a relational representation. In: Proceedings of the Workshop on Adaptability in Multi-Agent Systems at AORC (2003)
10. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco (1993)
11. Van Otterlo, M.: Efficient reinforcement learning using relational aggregation. In: Proceedings of the Sixth European Workshop on Reinforcement Learning (2003)
12. Srinivasan, A.: The Aleph Manual (2001)
13. Stone, P., Sutton, R.: Scaling reinforcement learning toward RoboCup soccer. In: Proceedings of the International Conference on Machine Learning (2001)
14. Sutton, R.: Integrated modeling and control based on reinforcement learning and dynamic programming. In: Advances in Neural Information Processing Systems, vol. 3 (1991)
15. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
16. Watkins, C.J.C.H.: Learning from Delayed Rewards. Ph.D. thesis, Cambridge University (1989)
17. Zettlemoyer, L.S., Pasula, H.M., Kaelbling, L.P.: Learning Planning Rules in Noisy Stochastic Worlds. In: Proceedings of the Twentieth Conference on Artificial Intelligence (2005)

An Inductive Learning System for XML Documents

Xiaobing Wu

CSIRO ICT Centre, Australia

Abstract. This paper presents a complete inductive learning system that aims to produce comprehensible theories for XML document classifications. The knowledge representation method is based on a higher-order logic formalism which is particularly suitable for structured-data learning systems. A systematic way of generating predicates is also given. The learning algorithm of the system is a modified standard decision-tree learning algorithm driven by predicate/recall breakeven point. Experimental results on XML version of Reuters dataset show that this system is able to produce comprehensible theories with high precision/recall breakeven point values.

Keywords: higher-order logic, knowledge representation, XML documents, precision-recall, decision-tree learning.

1 Introduction

XML (eXtensible Markup Language) documents are one of the most important sources of semistructured data. Semistructured data is data that has some structure, but the structure may not be rigid, or complete and generally the data does not conform to a fixed schema. Higher-order logic is particularly suitable for structured-data learning systems, as it is able to represent individuals with complex structures, precisely describe the hypothesis languages, and support the testing of hypotheses on individuals.

Decision-tree algorithms for learning are well-studied. In contrast to many other learning algorithms, such as neural networks and support vector machines, decision trees can learn theories that are comprehensive to users. A comprehensible theory could provide insight about the observations. Most existing decision-tree algorithms are based on accuracy heuristics. However, sometimes accuracy is not a good criterion for classification. Precision and recall are two trade-off criteria which are traditional standards for text document classification problems. Most XML documents have substantial free text content, so precision and recall are more appropriate criteria than accuracy for XML document classification.

This paper presents a novel inductive learning system for XML documents. Section 2 gives the representation method for XML documents using the higher-order logic formalism. Section 3 presents the decision-tree learning algorithm driven by precision and recall. Section 4 shows the experimental results on an XML version of the Reuters dataset.

H. Blockeel et al. (Eds.): ILP 2007, LNAI 4894, pp. 292–306, 2008.

2 Knowledge Representation for XML Documents

We adopt a typed higher-order logic as the knowledge representation formalism [6] for XML documents, as it is particularly suitable for representing individuals with complex structures. The higher-order logic is used in two essential ways here. First, individuals are represented as higher-order logic terms; second, predicates are constructed by composing transformations.

2.1 The Structure of an XML Document

This section is an overview of the general structure of XML documents.

An XML document has a nested structure. Each entity is enclosed by a start tag and an end tag. This structure is best described by an example. A simple but complete XML document is given in Figure 1. The first line of the document is the XML version declaration. The second line is the declaration of its DTD (Document Type Definition). Next comes the root element *bib* which has two sub-elements *book* describing information about two books. The first *book* has four further sub-elements *title*, *author*, *publisher* and *price*, which describe the first book. The first *book* also has an attribute *year* with a value of *2003*. The *author* further contains two sub-elements *last* and *first* which contains the last name and the first name of the author, respectively. The second *book* element is similar to the first one, but contains three *author* elements.

2.2 Representation of Individuals

A well-formed XML document is represented as a six-tuple.

$$type\ XML = XMLDecl \times Misclist \times DTD \times Misclist \times Element \times Misclist$$

Here, *XMLDecl* represents the XML declaration; *Misclist* represents a list of miscellaneous items such as comments, processing instructions, and spaces; *DTD* represents the document type declaration; and *Element* represents the root element of the document. All these six components are non-atomic type values and could be further defined. As an example, we give the representation of *Element* below.

The formal representation for an element is

$$\begin{gathered} data\ Element = Elem\ \ TagName\ \ Attributelist\ \ Contents \\ type\ \ TagName = String \\ type\ \ Attributelist = [Attribute] \\ type\ \ Contents = [Content] \end{gathered}$$

Here, type *Elem* is defined as a data constructor which has three arguments representing the element name, the attributes and the element contents, respectively. *TagName* is a synonym of type *String*. *Attributelist* and *Contents* is a list of attributes and a list of content, respectively.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE bib SYSTEM‘‘books.dtd’’>
<bib>
  <book year=‘‘2003’’>
    <title>Logic for Learning</title>
    <author>
       <last>Lloyd</last>
       <first>John</first></author>
    <publisher>Springer</publisher>
    <price>49.95</price>
  </book>
  <book year=‘‘2000’’>
    <title>Data on the Web</title>
    <author>
      <last>Abiteboul</last>
      <first>Serge</first></author>
    <author>
      <last>Buneman</last>
      <first>Peter</first></author>
    <author>
      <last>Suciu</last>
      <first>Dan</first></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
  </book>
</bib>
```

Fig. 1. An example XML document with an external DTD

An attribute is composed of the attribute name and attribute value, and is represented using a tuple as follows.

$$type\ Attribute = AttName \times AttValue$$

where *AttName* and *AttValue* are both synonym of *String*, written as follows.

$$type\ AttName = String$$
$$type\ AttValue = String$$

It is the element content that makes XML document nested and hierarchical. The content of an element could be another element, a piece of text, a reference to some entities, a CDATA section, a processing instruction or a comment. The content is formally represented as follows.

$$data\ Content = El\ Element \mid Tx\ CharData \mid Ref\ Reference \mid CD\ CData \mid ContentPI\ PI \mid ContentCom\ Comment$$

Here, *El*, *Tx*, and so on, are constructors of type *Content*. Constructor *El* needs an argument of type *Element* to construct data of type *Content*, and *Tx* needs an argument of type *CharData* to construct a type *Content*, and so on. Type *Element* has been introduced above. *CharData* is a synonym of a list of characters.

$$type\ CharData = [Char]$$

A complete representation for XML documents using this method can be found in [9].

2.3 Representation of Features

Here, features refer to predicates on the type of individuals. These predicates are constructed incrementally by composition of *transformations* using *predicate rewrite systems* [6]. Composition is handled by the composition function $\circ : (a \to b) \to (b \to c) \to (a \to c)$ defined by $((f \circ g)\ x) = (g\ (f\ x))$. A *transformation* f is a function having a signature of the form $f : (\varrho_1 \to \Omega) \to \cdots \to (\varrho_k \to \Omega) \to \mu \to \sigma$, where $\varrho_1, \ldots, \varrho_k$, σ and μ are all types, $k \geq 0$, and Ω is the type of the booleans.

Transformations on XML are classified into two categories: *generic transformations* and *data-specific transformations*. *Generic transformations* come straight from the XML document representation and are applicable to all well-formed XML documents. Some examples of generic transformations are given next.

$$\begin{aligned} projRootElement &: XML \to Element \\ projTagName &: Element \to TagName \\ projAttributes &: Element \to Attributes \\ projContents &: Element \to Contents \end{aligned}$$

Here, *projRootElement* projects onto the root element of an XML document. *projTagName*, *projAttributes* and *projContents* project an element onto its tag name, attributes and contents, respectively.

Another example of generic transformations on element attributes are given next.

$$\begin{aligned} listToSet &: Attributes \to \{Attribute\} \\ setExists_1 &: (Attribute \to \Omega) \to \{Attribute\} \to \Omega \end{aligned}$$

The above two transformations convert a list of attributes to a set of attributes and check whether the set has an attribute satisfying a predicate, respectively.

Data-specific transformations capture some specialized concepts that will be useful in a particular application. For example, a transformation $(=$ "*book*"$)$: $TagName \to \Omega$ is a data-specific transformation which checks whether a *TagName* is the string "book".

A predicate rewrite system is used to define and enumerate a set of predicates relevant to a particular application by using the transformations. A predicate rewrite system is a finite relation $\rightarrowtail$ on the set of predicates definable in the logic. A predicate rewrite has the form $p \rightarrowtail q$, where p and q are predicates and q is stronger than p. Given a predicate rewrite system, to generate a predicate search space for an application, one starts with an initial predicate p_0, usually the weakest one *top* (defined by $(top\ x) = True$ for all x), and generates all the predicates that can be obtained by a predicate derivation step from p_0, then all the predicates that can be obtained from those by a predicate derivation step, and so on. Now a simple predicate rewrite system is given as an example to illustrate how to generate a predicate search space via the predicate rewrite system.

$$
\begin{gathered}
top \rightarrowtail projRootElement \circ top \\
top \rightarrowtail \wedge_2\ (projTagName \circ top\ projContents \circ top) \\
top \rightarrowtail listToSet \circ top \\
top \rightarrowtail setExists_1\ (\wedge_2\ (isElement\ projContentElement \circ top)) \\
top \rightarrowtail (= \text{“}bib\text{”}) \\
top \rightarrowtail (= \text{“}book\text{”}) \\
top \rightarrowtail (= \text{“}author\text{”}) \\
top \rightarrowtail (= \text{“}price\text{”})
\end{gathered}
$$

The following is a path in the search space using the above predicate rewrite system with a start from the weakest predicate *top*.

$$
\begin{aligned}
&top \\
&projRootElement \circ top \\
&projRootElement \circ \wedge_2\ (projTagName \circ top\ projContents \circ top) \\
&projRootElement \circ \wedge_2\ (projTagName \circ (= \text{“}bib\text{”})\ projContents \circ top) \\
&projRootElement \circ \wedge_2\ (projTagName \circ (= \text{“}bib\text{”})\ projContents \circ \\
&\qquad listToSet \circ top) \\
&projRootElement \circ \wedge_2\ (projTagName \circ (= \text{“}bib\text{”})\ projContents \circ \\
&\qquad listToSet \circ setExists_1\ (\wedge_2 (isElement \\
&\qquad\qquad projContentElement \circ top))) \\
&projRootElement \circ \wedge_2\ (projTagName \circ (= \text{“}bib\text{”})\ projContents \circ \\
&\qquad listToSet \circ setExists_1\ (\wedge_2 (isElement \\
&\qquad\qquad projContentElement \circ \wedge_2\ (projTagName \circ \\
&\qquad\qquad\qquad (= \text{“}book\text{”})\ projContents \circ top)))) \\
&\qquad\qquad \ldots
\end{aligned}
$$

3 Precision/Recall-Driven Decision-Tree (PRDT) Algorithm

Having represented a well-formed XML document as a typed higher-order logic term, in this section, we will present a decision-tree learning algorithm based on the precision and recall criterion for XML document classifications.

3.1 Precision and Recall

Precision and recall were originally two statistical measures widely used in information retrieval. They have been borrowed to evaluate the performance of text classification [5,7,10] recently.

Precision Pr and recall Rc can be defined using TP, FP and FN as follows

$$Pr = \frac{TP}{TP + FP} \qquad Rc = \frac{TP}{TP + FN} \tag{1}$$

where TP is the number of documents correctly assigned to the positive class; FP, the number of documents incorrectly assigned to the positive class; FN, the number of documents incorrectly assigned to the negative class; and TN, the number of documents correctly assigned to the negative class.

Normally there is a trade-off between high precision and high recall. A good classifier should have both high precision and high recall. When precision and recall are equal (or very close), this point is called *precision/recall-breakeven point (BEP)* of the system. BEP is commonly used as a performance measure in text classification problems. By (1), we can see if FP and FN are equal (or very close), we get at BEP.

The *F-measure* was introduced by van Rijsbergen [8]. The most commonly used form of F-measure is when $\beta = 1$. In this case, the F-measure becomes F_1 measure which balances precision and recall. F_1 measure is defined as:

$$F_1 = \frac{2Pr \cdot Rc}{Pr + Rc} \tag{2}$$

From (2), we can see when precision and recall are equal (at BEP), $F_1 = Pr = Rc$. F_1 is maximized when precision and recall are equal or close (at BEP). Hence, if we maximise F_1, we can get the BEP value which is equal (or close) to the maximized F_1. By using F_1 measure, we turn the two measurements into a single measurement which is easier to work with in an optimisation system.

3.2 Structured Feature Selection

Unlike unstructured text documents, the same word appearing in different elements of an XML document could have different meaning or influence on

document classification. Different elements could contain irrelevant text content. Therefore, it is not suitable to gather all the text in a XML document and conduct feature selection. we propose to build n independent corpora of text by analysing the DTD, where n is the number of elements in the DTD. The text in an element of an XML document is collected and formed a new text document with the same name as the original XML document, and stored in a corpus corresponding to this element. Thus, all the documents in the same corpus come from the same *element* and are ready to do feature selection.

In text learning, a *feature* is defined for each word in the training set. Feature selection is commonly used when learning on text, as text documents often have thousands of features. Some of them are not relevant or not beneficial for the performance of the text learning. Removing these features can greatly improve the learning speed and the classification accuracy. *Feature selection* aims to select a minimal subset of features which still contain enough features for classification. The commonly used approach for feature selection in text learning is to use a fixed measure method to evaluate and score all the features and then sort them by score.

Mutual information [5,3,11] is a commonly used criterion for feature selection in text categorization, and this method is adopted in this work. It measures the mutual information between a word and a class. In information theory, mutual information is defined as the reduction in the entropy of a random variable due to introducing another random variable.

Like in traditional text learning, we introduce a weighting scheme into our text features. The occurrences of some features in a document provides a better indication of the content of the document than other features. The locality of a feature decides its weight in classification, i.e., the localised feature that occurs frequently in only a few documents is more informative, but the global feature that occurs frequently in most of the documents is not informative. We adopt the *TFIDF* measure which is an important and commonly used weighting scheme. It combines term frequencies (TF) with inverse document frequencies (IDF). *Term Frequency* $TF(t, d)$ is defined as the number of times a term t occurs in a document d. *Document Frequency* $DF(t)$ is the number of documents in which term t occurs at least once. The *inverse document frequency* $IDF(t)$ can be calculated from $DF(t)$ by

$$IDF(t) = log(\frac{N}{DF(t)})$$

where N is the total number of documents.

$TFIDF(t, d)$ is defined as

$$TFIDF(t, d) = TF(t, d) \times IDF(t)$$

The idea behind the *TFIDF* measure is that a term t is more important as a feature for document d if it appears more frequently in d and appears in fewer documents overall.

Under this model, a text document d is represented as

$$d =< (t_1, tfidf(t_1)), \ldots, (t_n, tfidf(t_n)) >$$

where $t_1, \ldots, t_n$ are the IDs of the features in the feature subset which appear in document d.

3.3 Node Selection

Starting from a single root node containing all the training examples (some individuals with annotated classifications), the decision-tree algorithm iteratively makes a binary split at a selected node in the existing tree. In our PRDT algorithm, we use a novel node selection method to control the system to work towards reducing the difference between the precision and recall. Next, we give some theoretical analysis for our node selection method.

First, we define the TP, FP and FN values for the node associated with $\mathcal{E}$, where $\mathcal{E}$ is a (non-empty) set of examples.

Definition 1. *Let $\mathcal{E}$ be a set of examples. We define:*

1. $TP(\mathcal{E}) = n_+^{\mathcal{E}},\ FP(\mathcal{E}) = n_-^{\mathcal{E}},\ FN(\mathcal{E}) = 0 \qquad if\ n_+^{\mathcal{E}} \geq n_-^{\mathcal{E}} \geq 0$
2. $TP(\mathcal{E}) = 0,\ FP(\mathcal{E}) = 0,\ FN(\mathcal{E}) = n_+^{\mathcal{E}} \qquad if\ n_-^{\mathcal{E}} > n_+^{\mathcal{E}} \geq 0.$

In the above definition, $n_+^{\mathcal{E}}$ and $n_-^{\mathcal{E}}$ denote the number of positive examples and negative examples in $\mathcal{E}$, respectively. A decision-tree node is called a *FP-type* node if $FP(\mathcal{E}) \geq 0$, otherwise an *FN-type* node. It is not hard to see that an FP-type set of examples is a positive majority set of examples and an FN-type set of examples is a negative majority set of examples.

For a decision tree T, to balance the $FP(T)$ and $FN(T)$, our node selection method is

1. select a FP-type leaf node of T, if $FP(T) > FN(T)$, or
2. select a FN-type leaf node of T, if $FP(T) < FN(T)$.

Next, we give a proposition to support our node selection method above. Let $\mathcal{P} = \{\mathcal{E}_1, \mathcal{E}_2\}$ be a partition of a set of examples $\mathcal{E}$. Proposition 1 shows that $FP(\mathcal{P})$ will decrease and $FN(\mathcal{P})$ will increase if $\mathcal{E}$ is FP-type. Similarly, $FN(\mathcal{P})$ will decrease and $FP(\mathcal{P})$ will increase if $\mathcal{E}$ is FN-type.

Proposition 1. *Let $\mathcal{E}$ be a set of examples and $\mathcal{P} = (\mathcal{E}_1, \mathcal{E}_2)$ a partition of $\mathcal{E}$.*

1. *If $\mathcal{E}$ is FP-type, then $FP(\mathcal{E}) \geq FP(\mathcal{P})$, $FN(\mathcal{E}) \leq FN(\mathcal{P})$.*
2. *If $\mathcal{E}$ is FN-type, then $FP(\mathcal{E}) \leq FP(\mathcal{P})$, $FN(\mathcal{E}) \geq FN(\mathcal{P})$.*

Proof. 1. $\mathcal{E}$ is a FP-type, so it is a set of positive majority examples. There are three cases to consider: $\mathcal{E}_1$ and $\mathcal{E}_2$ are both FP-type, both FN-type, or one FP-type and one FN-type.

(a) Suppose $\mathcal{E}_1$ and $\mathcal{E}_2$ are both FP-type. $FP(\mathcal{P}) = n_-^{\mathcal{E}_1} + n_-^{\mathcal{E}_2} = n_-(\mathcal{E})$. Thus, $FP(\mathcal{E}) = FP(\mathcal{P})$. $FN(\mathcal{E})$, $FN(\mathcal{E}_1)$ and $FN(\mathcal{E}_2)$ are all 0. Thus, $FN(\mathcal{E}) = FN(\mathcal{P})$. Now the first case has been proved.

(b) Suppose $\mathcal{E}_1$ and $\mathcal{E}_2$ are both FN-type. Partition $\mathcal{P}$ partitions a positive majority set of examples, $\mathcal{E}$, into two negative majority sets of examples $\mathcal{E}_1$ and $\mathcal{E}_2$. This case could not happen.

(c) Suppose $\mathcal{E}_1$ and $\mathcal{E}_2$ are one FP-type and one FN-type. Without loss of generality, we suppose $\mathcal{E}_1$ is FP-type and $\mathcal{E}_2$ is FN-type. Thus, $n_+^{\mathcal{E}_1} \geq 0$, $n_-^{\mathcal{E}_2} > 0$, $FP(\mathcal{E}_1) = n_-^{\mathcal{E}_1}$, and $FP(\mathcal{E}_2) = 0$. $FP(\mathcal{P}) = n_-^{\mathcal{E}_1} < n_-^{\mathcal{E}_1} + n_-^{\mathcal{E}_2} = FP(\mathcal{E})$. $FN(\mathcal{E}_2) = n_-^{\mathcal{E}_2} > 0$. $FN(\mathcal{P}) = FN(\mathcal{E}_1) + FN(\mathcal{E}_2) = n_-^{\mathcal{E}_2}$, while $FN(\mathcal{E}) = 0$. Thus, $FN(\mathcal{E}) > FN(\mathcal{P})$.

2. Similar to the proof of 1.

Proposition 1 states that the *FP* value will decrease or not change, and the *FN* value will increase or not change by partitioning a FP-type set of examples. Similarly, the *FP* value will increase or not change, and the *FN* value will decrease or not change by partitioning a FN-type set of examples. As noted in Case (c) of the Part 1 of the proof, the *FP* (*FN*) value will strictly decrease (increase) by splitting a FP-type set of examples into two different types of sets of examples. In the same way, the *FP* (*FN*) value will strictly increase (decrease) by splitting a FN-type set of examples into two different types of sets of examples.

3.4 The Precision/Recall-Driven Decision-Tree Algorithm

In this section, the Precision/Recall-driven Decision-Tree (PRDT) algorithm will be described. (See Figures 2 and 3.)

Figure 2 is the PRDT algorithm. The goal of this algorithm is to find a tree that has the best precision-recall breakeven point value. Starting from a single node which is composed of the training data, the algorithm works towards two goals at the same time: looking for the point where the global precision and recall are equal and improving the F_1 measure. The first goal is achieved by selecting the node which can most balance the two values, that is, the leaf with the largest *FP* when $FP(T) \geq FN(T)$ and the leaf with the largest *FN* otherwise. The second goal is achieved by finding a predicate to split this node which can best improve $F_1(T)$. If the partition made to the selected node fails to improve $F_1(T)$, the leaf node with the next largest *FP* or *FN* will be chosen, and this procedure will continue until all the leaf nodes with positive *FP* or *FN* are tested and $F_1(T)$ fails to improve.

The training examples and a predicate rewrite system are input into the PRDT algorithm. The tree T is initialised as a single node containing all the training examples. The algorithm then enters an iteration where T is kept splitting. At the start of each iteration, i.e., when a new split is going to be made, three kinds of leaves are managed in three different sets: $leaves_{FP}$, $leaves_{FN}$ and $leaves_F$, which store those unseen leaves that have positive FP, positive FN and positive FP or FN, respectively. $Leaves(T)$ returns the set of leaves of tree T. The algorithm then enters an inner iteration which selects a leaf and splits

it. The leaf to be selected for splitting should mostly balance the $FP(T)$ and $FN(T)$. If $FP(T) > FN(T)$, then the leaf with the biggest FP is selected; if $FP(T) < FN(T)$, then the leaf with the biggest FN is selected; otherwise if FP and FN are already balanced but non-zero, then the leaf with the biggest FP or FN is selected. However, if $F_1(T)$ could not be improved by splitting the selected leaf, then this splitting should be given up and the next leaf that satisfies the corresponding conditions should be considered. This iteration continues until $F_1(T)$ improves by splitting the selected leaf or there is no leaf to select in the corresponding leaf set.

In Figure 2, $T(l, \mathcal{P})$ denotes the new tree obtained by splitting leaf l with partition $\mathcal{P}$.

function $PRDT(\mathcal{E}, \rightarrowtail)$; **returns**: a decision tree;
inputs: $\mathcal{E}$, a set of examples;
$\rightarrowtail$, a predicate rewrite system;

$T :=$ single node (with examples $\mathcal{E}$);
finished := false;
while not *finished* **do**
  $leaves_{FP} := \{l | FP(l) > 0 \wedge l \in Leaves(T)\}$;
  $leaves_{FN} := \{l | FN(l) > 0 \wedge l \in Leaves(T)\}$;
  $leaves_F := leaves_{FP} \cup leaves_{FN}$;
  while true **do**
    if $FP(T) \geq FN(T) \wedge leaves_{FP} \neq \phi$ **then**
      $l := argmax_{l \in leaves_{FP}} FP(l)$;
      $leaves_{FP} := leaves_{FP} \setminus \{l\}$;
    else if $FP(T) < FN(T) \wedge leaves_{FN} \neq \phi$ **then**
      $l := argmax_{l \in leaves_{FN}} FN(l)$;
      $leaves_{FN} := leaves_{FN} \setminus \{l\}$;
    else if $FP(T) = FN(T) \neq 0 \wedge leaves_F \neq \phi$
      $l := argmax_{l \in leaves_F} (FP(l) + FN(l))$;
      $leaves_F := leaves_F \setminus \{l\}$;
    else
      finished := true;
      break;
    $p :=$ *Predicate*($TP(T)$,$FP(T)$,$FN(T)$,$\mathcal{E}_l$,$\rightarrowtail$);
    $\mathcal{P} :=$ partition of $\mathcal{E}_l$ induced by p;
    if $F_1(T(l, \mathcal{P})) > F_1(T)$ **then**
      $T := T(l, \mathcal{P})$;
      break;

label each leaf node of T by its majority class;
return T;

Fig. 2. Decision-tree algorithm based on the precision/recall heuristic

The function *Predicate* in the PRDT algorithm is shown in Figure 3. The predicate output by this algorithm is the one that could best improve the global F_1 measure of the tree. The TP, FP and FN of the current tree T, the set of training examples in the selected leaf and the predicate rewrite system are input into function *Predicate*. An openlist is used to keep all the candidate predicates. Variable *predicate* is used to keep the current best predicate, and *bestScore* is used to keep the F_1 of the current best predicate. Initially, the *openList* contains only the weakest predicate *top*, and the *bestScore* is set to be the F_1 of the current tree T. The algorithm then enters an iteration. In each iteration, the first candidate predicate is drawn from the openlist and a sub-search space is generated from this predicate. Each predicate q in this sub-search space is tested by creating a new partition $\mathcal{P}$ using it. The F_1 of the tree, after adding the new partition $\mathcal{P}$, is computed using the updated TP, FP and FN. If F_1 is higher than the current *bestScore*, the best predicate and the best score are set as q and its corresponding F_1, respectively. Predicate q is also inserted into the openlist as a candidate for further sub-search space. The iteration terminates when the openlist is empty. Finally the predicate that most improves the F_1 of the tree by splitting the selected leaf is returned.

Note that function *predicate* may not terminate if the search space defined by $\rightarrowtail$ is infinite. In this case, a non-negative parameter *cutout* can be set to stop the searching if the algorithm investigates *cutout* predicates without finding a better predicate than the current best predicate. Every time a new predicate is found, the *cutout* parameter is reset to the initial value.

4 Experiments

In this section, experiments on a real-world dataset, the XML version of Reuters dataset, is reported. Reuters dataset is a traditional test collection for text categorization.

4.1 The Dataset

Reuters-21578[1] is a collection of newswire stories in Reuters newswire in 1987. It is a commonly used test collection for text classification [4,11,5,1]. The original collection consists of 22 SGML data files. Each of the first 21 files contain 1000 articles, while the last contains 578 articles.

The 21578 documents are assigned to 135 categories according to their TOPICS attributes. The "ModApte" split, which leads to a corpus of 9603 training documents and 3299 test documents, is used in all experiments here. The number of documents in each category varies widely, ranging from "Earn" which contains 3964 documents to "Castor-oil" which contains only one document. However, the ten most frequent categories account for 75% of the training instances. These ten categories are often used in the experiments of text categorization.

[1] http://www.daviddlewis.com/resources/testcollections/reuters21578/

In the preprocessing, each article in each SGML file was transformed to an XML document. In each XML document, there are altogether about 12 elements. The categories of an XML document are set by element `TOPICS` which are determined by the text content of the document instead of other factors. The text content of the article is stored in element *TEXT* which has four sub-elements: *TITLE*, *BODY*, *AUTHOR* and *DATELINE*. *TITLE* stores the title of the story and *BODY* stores the main text of the story. The background knowledge tells us that the hypothesis should be with the element *TEXT*, to be more specific, should be with element *TITLE* and *BODY*. This information helps us build the predicate rewrite system.

The text contents in the XML documents are preprocessed using the structured feature selection method described in Section 3.2. The text content in an element is collected and formed a new text document and stored in a

function $Predicate(TP, FP, FN, \mathcal{E}, \rightarrowtail)$ **returns** a predicate;

inputs: TP, TP of the current existing tree;
FP, FP of the current existing tree;
FN, FN of the current existing tree;
$\mathcal{E}$, a set of examples;
$\rightarrowtail$, a predicate rewrite system;

$openList := [top]$;
$predicate := top$;
calculate Pr and Rc of the current existing tree;
$bestScore := \frac{2Pr \times Rc}{Pr+Rc}$;
while $openList \neq []$

$p := \text{head}(openList)$;
$openList := tail(openList)$;
for each LR redex r via $r \rightarrowtail b$, for some b, in p **do**
$q := p[r/b]$;
if q is regular **then**
$\mathcal{P} :=$ partition of $\mathcal{E}$ induced by q;
update TP, FP, FN;
update Pr, Rc;
$F_1 := \frac{2Pr \times Rc}{Pr+Rc}$;
if $F_1 > bestScore$ **then**
$predicate := q$;
$bestScore := F_1$;
$openList := Insert(q, openList)$;

return $predicate$;

Fig. 3. Algorithm for finding a predicate to split a node

corpus corresponding to this element. Feature selection is done independently for each corpus. The procedure of the feature selection includes stop words removal, stemming, feature selection and feature weighting.

4.2 Experimental Results

Part of the predicate rewrite system used in our experiments are as follows.

$$top \rightarrowtail projRootElement \circ top$$
$$top \rightarrowtail \wedge_2\ (projTagName \circ top\ projContents \circ top)$$
$$top \rightarrowtail listToSet \circ top$$
$$top \rightarrowtail setExists_1\ (\wedge_2\ (isElement\ projContentElement \circ top))$$
$$top \rightarrowtail setExists_1\ (\wedge_2\ (isFeature\ projContentFeature \circ top))$$
$$top \rightarrowtail setExists_1\ (\wedge_2\ (proFeatureId \circ top\ projFeatureWeight \circ top))$$
$$top \rightarrowtail (= REUTERS)$$
$$top \rightarrowtail (= i)$$
$$top \rightarrowtail (\geq x)$$
$$top \rightarrowtail (\leq y)$$

In the above predicate rewrite system, i is an integer representing the text feature number, and x $(0 \leq x \leq 1)$ and y $(0 \leq y \leq 1)$ represents the feature weights.

One binary decision tree was built for each of the 10 most frequent classes. Table 1 gives the precision/recall-breakeven point value on the ten most frequent categories and their micro-average for five learning algorithms. The results for Findsim, NBayes, BayesNets and LinearSVM are reported in [2]. No published

Table 1. Precision/recall-breakeven point on the ten most frequent Reuters categories and their micro-average

	Findsim	NBayes	BayesNets	PRDT	LinearSVM
earn	92.9%	95.9%	95.8%	96.4%	98%
acq	64.7%	87.8%	88.3%	85.8%	93.6%
money-fx	46.7%	56.6%	58.8%	66.5%	74.5%
grain	67.5%	78.8%	81.4%	93.9%	94.6%
crude	70.1%	79.5%	79.6%	84.9%	88.9%
trade	65.1%	63.9%	69.0%	71.2%	75.9%
interest	63.4%	64.9%	71.3%	74.2%	77.7%
ship	49.2%	85.4%	85.4%	76.6%	85.6%
wheat	68.9%	69.7%	82.7%	92.1%	90.3%
corn	48.2%	65.3%	76.4%	90.4%	90.3%
microave	64.6%	81.5 %	85.0 %	88.0 %	92.0 %

results are available for the distance between the precision and recall on the breakeven point for those algorithms. Table 2 summarises the precision and recall on the test set for each of the ten classes on the test set for PRDT. The two values of most classes are very close (around 5%), and the average difference of the two values is 8.1%.

Table 2. The precision and recall value on the ten most frequent categories of Reuters and their average

	earn	acq	money	grain	crude	trade	intst	ship	wht	corn	avr
Pr	95.5	89.6	72.2	91.1	86.7	67.2	79.6	84.7	88.0	84.4	89.2
Rc	97.3	81.9	60.9	96.6	83.1	75.2	68.7	68.5	96.2	96.4	86.8

From Table 1 and Table 2, we can see that the PRDT algorithm performs well on producing hypotheses with high BEP values on all 10 classes. Another important point is that the PRDT algorithm provides comprehensible hypotheses while most of other algorithms in Table 1 do not. The hypothesis produced by PRDT algorithm contains information of the structure that is comprehensible to people. Though the text features that appear in the hypothesis are not comprehensible at first sight, it is easy to transform the features IDs to their original terms by searching the feature-ID table.

Here, an example is given to show the comprehensibility of the hypothesis produced by our system. The following one-split decision tree gives a binary classifier for category "trade".

$$
\begin{aligned}
&trade\ m = \\
&\quad if\ projRootElement \circ \wedge_2\ (projTagName \circ (= REUTERS)\ projContents \circ \\
&\qquad listToSet \circ setExists_1\ (\wedge_2 (isElement\ projContentElement \circ \\
&\qquad \wedge_2 (projTagName \circ (= TEXT)\ projContents \circ listToSet \circ \\
&\qquad setExists_1 (\wedge_2 (isElement\ projContentElement \circ \\
&\qquad \wedge_2 (projTagName \circ (= TITLE)\ projContents \circ \\
&\qquad listToSet \circ setExists_1 (isFeature\ projContentFeature \circ setExists_1 (\\
&\qquad \wedge_2 (projFeatureId \circ (= 89)\ projFeatureWeight \circ top)))))))))\ m \\
&\quad then\ \top \\
&\quad else\ \bot
\end{aligned}
$$

This theory clearly states the structure of the XML documents belonging to "trade" class as follows.

"An XML document belongs to class trade iff (i) the root element of the document is "REUTERS" and (ii) it contains an element named "TEXT" and (iii) element "TEXT" contains an element named "TITLE" and (iv) element "TITLE" contains the feature No.89 which represents word "budget"."

5 Conclusion

This paper presents a novel inductive learning system that aims to produce comprehensible hypothesis for XML document classification. The knowledge representation method is based on a higher-order logic formalism which is suitable for representing individuals with complex structures. A systematic way for generating predicates is given by using predicate rewrite systems. The learning algorithm of our system is a decision-tree learning algorithm driven by precision and recall. The algorithm works towards two goals at the same time: decreasing the difference between the precision and recall and improve the F_1 value. Experimental results show that the PRDT algorithm performs very well on XML data classification, and its ability to proving comprehensible theories make it more distinctive.

References

1. Dagan, I., Karov, Y., Roth, D.: Mistake-driven learning in text categorization. In: Proceedings of the Second Conference on Empirical Methods in Natural Language Processing, AAAI Press, Menlo Park (1997)
2. Dumais, S., Platt, J., Heckerman, D., Sahami, M.: Inductive learning algorithms and representations for text categorization. In: Proceedings of the Seventh International Conference on Information and Knowledge Management, pp. 148–155 (1998)
3. Dumais, S.T., Chen, H.: Hierarchical classification of web content. In: Proceedings of ACM-SIGIR International Conference on Research and Development in Information Retrieval, Athens, pp. 256–263 (2000)
4. Joachims, T.: A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In: Proceedings of ICML 1997, 14th International Conference on Machine Learning (1997)
5. Lewis, D., Ringuette, M.: A comparison of two learning algorithms for text categorization. In: Proceedings of SDAIR 1994, 3rd Annual Symposium on Document Analysis and Information Retrieval (1994)
6. Lloyd, J.W.: Logic for Learning: Learning Comprehensible Theories from Structured Data. Springer, Heidelberg (2003)
7. Sebastiani, F.: A tutorial on automated text categorisation. In: Proceedings of ASAI 1999, First Argentinian Symposium on Artificial Intelligence, Buenos Aires, AR, pp. 7–35 (1999)
8. van Rijsbergen, C.J.: Information Retrieval. Butterworths, London (1979)
9. Wu, X.: Knowledge Representation and Learning For Semistructured Data. PhD thesis, The Australian National University (2006)
10. Yang, Y.: An evaluation of statistical approaches to text categorization. ACM Transactions on Information Systems 12(3), 296–333 (1998)
11. Yang, Y., Pedersen, J.O.: A comparative study on feature selection in text categorization. In: Proceedings of ICML 1997, 14th International Conference on Machine Learning, Nashville, TX, Fisher, D.H. (eds.). pp. 412–420 (1997)

Author Index