Lecture Notes in Artificial Intelligence 2583

Subseries of Lecture Notes in Computer Science
Edited by J. G. Carbonell and J. Siekmann

Lecture Notes in Computer Science
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Stan Matwin Claude Sammut (Eds.)

Inductive Logic Programming

12th International Conference, ILP 2002
Sydney, Australia, July 9-11, 2002
Revised Papers

Springer

Series Editors

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

Volume Editors

Stan Matwin
University of Ottawa, School of Information Technology and Engineering
800 King Edward Ave., Ottawa, ON, K1N 6N5 Canada
E-mail: stan@site.uottawa.ca

Claude Sammut
University of New South Wales, School of Computer Science and Engineering
Sydney, NSW 2052, Australia
E-mail: claude@cse.unsw.edu.au

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie; detailed bibliographic data is available in the Internet at <http://dnd.ddb.de>.

CR Subject Classification (1998): I.2.3, I.2, D.1.6, F.4.1, F.2.2, H.2.8

ISSN 0302-9743
ISBN 3-540-00567-6 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York,
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergraphik
Printed on acid-free paper SPIN: 10872409 06/3142 5 4 3 2 1 0

Preface

The Twelfth International Conference on Inductive Logic Programming was held in Sydney, Australia, July 9–11, 2002. The conference was colocated with two other events, the Nineteenth International Conference on Machine Learning (ICML 2002) and the Fifteenth Annual Conference on Computational Learning Theory (COLT 2002).

Started in 1991, Inductive Logic Programming is the leading annual forum for researchers working in Inductive Logic Programming and Relational Learning.

Continuing a series of international conferences devoted to Inductive Logic Programming and Relational Learning, ILP 2002 was the central event in 2002 for researchers interested in learning relational knowledge from examples.

The Program Committee, following a resolution of the Community Meeting in Strasbourg in September 2001, took upon itself the issue of the possible change of the name of the conference. Following an extended e-mail discussion, a number of proposed names were subjected to a vote. In the first stage of the vote, two names were retained for the second vote. The two names were: Inductive Logic Programming, and Relational Learning. It had been decided that a 60% vote would be needed to change the name; the result of the vote was 57% in favor of the name Relational Learning. Consequently, the name Inductive Logic Programming was kept.

In order to facilitate submissions, the Program Committee decided to extend the deadline and at the same time publish the proceedings after the conference (the extended deadline was cutting into the time needed by the publisher to produce the proceedings from the camera-ready copy). At the extended deadline there were 45 submissions to the conference, which resulted in 24 accepted papers, of which two withdrew and 22 were presented.

The conference program consisted of two invited talks, the presentations of accepted technical papers, a Work-in-Progress session, and a panel. The invited talk by John Shawe-Taylor introduced the concept of the Set Covering Machine and its relations to ILP. The second invited talk, by Sašo Džeroski, joint with ICML, reviewed the goals of ILP and showed how they fruitfully support Computational Scientific Discovery. There was a Work-in-Progress session with 10 papers, whose abstracts were published in a separate booklet. Furthermore, there was a panel "If I had to start a Ph.D. in ILP today, what would my topic be?", chaired by S. Matwin, with the following panelists: I. Bratko, P. Flach, J. Lloyd, S. Muggleton, C. Rouveirol, P. Tadepalli, and L. Saitta.

The ILP 2002 technical program, as with previous editions of the conference, covered a broad range of topics, from implementation techniques to applications, and from kernels for structured data to associations for relational data.

ILP 2002 would not have been possible without the generous support of ILPNet2 to European participants. This is kindly acknowledged. Several volunteers assisted the Program Chairs in different phases of the organization of

the conference. We are very grateful to Erick Alphonse from Université Paris XI for his help in organizing the distribution of papers to reviewers, to Svetlana Kiritchenko from the University of Ottawa for her help in producing the camera-ready version of these proceedings, and to Michael Bain from the University of New South Wales for his help in local arrangements.

November 2002

Stan Matwin
Claude Sammut

Organization

Program Chairs: Stan Matwin, University of Ottawa, Canada
Claude Sammut, University of New South Wales, Australia

Program Committee

H. Blockeel (Belgium)
J.F. Boulicaut (France)
I. Bratko (Slovenia)
J. Cussens (UK)
T. Dietterich (USA)
S. Džeroski (Slovenia)
F. Esposito (Italy)
P. Flach (UK)
J. Fürnkranz (Austria)
K. Furukawa (Japan)
L. Getoor (USA)
L. Holder (USA)
T. Horvath (Germany)
D. Jensen (USA)
R. Khardon (USA)
J.-U. Kietz (Switzerland)
S. Kramer (Germany)
N. Lavrač (Slovenia)
R. Mooney (USA)
S. Muggleton (UK)
D. Page (USA)
B. Pfahringer (NZ)
L. De Raedt (Germany)
D. Roth (USA)
C. Rouveirol (France)
L. Saitta (Italy)
M. Sebag (France)
A. Srinivasan (UK)
P. Tadepalli (USA)
S. Wrobel (Germany)
A. Yamamoto (Japan)
G. Zaverucha (Brazil)
J.-D. Zucker (France)

Table of Contents

Contributed Papers

Invited Talk Abstracts

Propositionalization for Clustering Symbolic Relational Descriptions

Isabelle Bournaud[1], Mélanie Courtine[2], and Jean-Daniel Zucker[2]

[1] LRI, Université Paris-Sud, Av. du Général de Gaulle, 91405 Orsay Cedex, France
Isabelle.Bournaud@lri.fr
[2] LIP6 - Pole IA, Université Paris VI, 4 place Jussieu, 75252 Paris Cedex, France
{Melanie.Courtine,Jean-Daniel.Zucker}@lip6.fr

Abstract. Propositionalization has recently received much attention in the ILP community as a mean to learn efficiently non-determinate concepts using adapted propositional algorithms. This paper proposes to extend such an approach to unsupervised learning from symbolic relational description. To help deal with the known combinatorial explosion of the number of possible clusters and the size of their descriptions, we suggest an approach that gradually increases the expressivity of the relational language used to describe the classes. At each level, only the initial object descriptions that could benefit from such an enriched generalization language are propositionalized. This latter representation allows us to use an efficient propositional clustering algorithm. This approach is implemented in the CAC system. Experiments on a large Chinese character database show the interest of using KIDS to cluster relational descriptions and pinpoint current problems for analyzing relational classifications.

1 Introduction

Automatically building conceptual classifications has been the subject of much research during the last twenty years (Michalski 81, Gennari 89, De Raedt 01). It consists in searching for similarities among objects that are not pre-classified, and structuring them into a hierarchy in which *similar* objects are clustered. What makes this problem different from the traditional clustering problem (Diday 71, Cheeseman 88) is that not only objects have to be clustered but an explicit description of the clusters must also be given. This explains why this field is called *conceptual* clustering.

Early approaches in conceptual clustering have defined this task as the search for *the* classification that would best predict unknown features of new objects (Fisher 85, Gennari 89, Fisher 96). Many of them can be modelled using a two-step process. In the first step, a similarity measure is used to build clusters that satisfy the above-mentioned criteria. In the second step, a description of the clusters is learned using supervised learning techniques. The methods developed have proved their interest in various application domains (Michalski 81, Fisher 87, Gennari 89,Ketterlin 95, De Raedt 01). More recent research concerns the construction of classifications that *organize knowledge* (Mineau 90, Carpineto 93).

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 1–16, 2003.

In this task, the goal is not to build a *single* hierarchy of objects but to combine different hierarchies into a structure called the Generalization Space. Such a structure takes into account the various dimensions along which two objects might be similar. In this task, unlike in conceptual clustering, it is necessary to choose a distance measure that aggregates all the dimensions of the descriptions such as Euclidian distance, for instance. On the contrary, a particular distance may be chosen for each individual dimension. In this way, it is possible to allow to accommodate for symbolic attributes that typically causes problem when integrated into a global numerical distance (Wilson 00). All existing approaches base their distance on the syntactical similarity between descriptions. Given this set of distances, objects are clustered together whenever one or more of the distances between their attribute values is not null.

Efficient algorithms have been proposed for organizing symbolic data described by a set of <attribute, value> pairs (e.g. MSG (Mineau 90)) and for taking into account domain knowledge (e.g. COING (Bournaud 97)). Although representing the objects to cluster as oriented labeled graphs, all aforementioned approaches based on a Generalization Space do not take into account the relational aspect of descriptions, more specifically the *non-determinate* descriptions. What these algorithms do can be described as a kind of *propositionalization* (De Raedt 98, Zucker & Ganascia, 98, Lavrac 01), each graph being reformulated into a bag of independant arcs. Our research concerns the organization of relational data, i.e. data that are represented in expressive formalisms (first-order logic, description logic, conceptual graphs, Datalog clauses, etc.). Unfortunately, the generalization of relational descriptions needed to build a Generalization Space requires matching graphs and leads to a known combinatorial explosion. To help deal with the complexity of this problem, we suggest an approach that gradually increases the potential complexity of the descriptions. In its first step, KIDS uses a propositional language and builds a propositional Generalization Space using the COING algorithm (Bournaud 97). KIDS gradually enriches this space thanks to a propositionalization that is based on more and more complex pattern. This approach, namely gradually increasing the structure of the matching, is inspired by the supervised learning systems REMO (Zucker 96) and REPART (Zucker & Ganascia, 98).

The next section briefly presents COING, a practical approach to build a Generalization Space of relational descriptions by performing a propositionalization using *an arc* of the graph as reformulation pattern. This endows COING with a quadratic complexity but the Generalization Space does not contain any relational similarities. Section 3 introduces the notion of abstract relations and their use for the reformulation of relational descriptions. Section 4 describes the principle of the KIDS approach to build a true relational Generalization Space. In the next section, we evaluate in practice KIDS on a Chinese character database by analyzing its complexity in terms of speed and number of classes build. Section 6 concludes and outlines directions for future research.

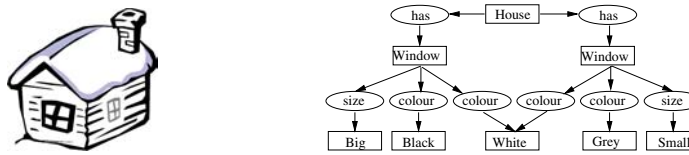


Fig. 1. A house and its description as a graph.

2 Organizing Propositionalized Relational Data

2.1 A Graphical Representation of Relational Data

We are concerned with the task of organizing knowledge represented as structured objects, that may be decomposed into several parts, and these are then linked together thanks to various relations (for example a *part-of* relation). Our approach is based on the use of a graph formalism to describe relational data. The choice of a graph formalism (such as conceptual graphs (Sowa 84, Chein 92), or Haussler graphs (Haussler 89)) rather than a logical one (Horn clauses (Dehaspe 98a, Flach 01)) is that it is easy to formulate the principle of our approach, based on the notion of *graph reformulation*, into a set of subgraphs. However, as mentioned in section 3.3, the approach is not dependent on the formalism used and may be transposed to a logical formalism. We now consider that a *description* of an object or a relational data is an oriented labelled graph whose vertices are of two types: **concept vertices** and **relation vertices**. Moreover, only binary relations are used in a description. It should be noted that the graph formalism used, and its graphical representation, is a sub-set of the conceptual graph formalism (Sowa 84,Chein 92) and is limited to binary relations. Figure 1 presents the example of the description of a house. This example is used throughout the article to illustrate the algorithms presented.

In this figure, the concept label *Window* appears twice, whereas the concept label *White* appears only once. This is due to the fact that it is necessary to distinguish two instances of *Window* (it is not the same window), but two instances of *White*. A graph is composed of a set of arcs, defined as a triplet: $[concept_s] \rightarrow (relation) \rightarrow [concept_t]$, where *(relation)* corresponds to a relation between $[concept_s]$ and $[concept_t]$.

2.2 Organizing Data in a Generalization Space

Given a set of object descriptions and a generalization language, the Generalization Space (GS) is a pruned inheritance concept lattice where each node is the least general generalization of the descriptions of the objects covered by the node. A pruned inheritance concept lattice is derived from the associated concept lattice (Wille 82) in eliminating the inherited part of the descriptions and nodes with empty descriptions. In other words, a node n_i of the GS is a pair $(c_i,\ d_i)$. The element c_i, called the coverage of n_i, is the set of objects covered by n_i; and d_i, called the description of n_i, corresponds to the common features

(least general generalization) of the objects of c_i. GS nodes are partially ordered by a subsumption relation between concepts. Given a node n_i with coverage c_i, its ancestors are all the nodes n_j, such that $c_j \supset c_i$. Indeed, GS nodes inherit the descriptions of the nodes which are more general. The Generalization Space may also be defined by the two isomorphic lattices: the Galois lattice of concept descriptions (partially ordered by the subsumption relation) and the object lattice (partially ordered by the inclusion relation) (Liquiere 98). In the case of a propositional generalization language, the least general generalization of a set of descriptions is *unique*. In the other case, a node description contains all the least general generalizations - w.r.t. the generalization language considered - of the set of descriptions.

Figures 2 and 4 present two different Generalization Spaces of the same objects Their differences lie in the expressiveness of the generalization language used to build the GS. Given a set of object descriptions, depending on the language chosen to describe the generalizations (the GS node descriptions), the nodes of the associated GS will not be the same: some nodes may appear only in one of the GS (cf. *n'3* in figure 4) and nodes having the same coverage may have a more or less general description (cf. *n2* in figure 2 and *n'2* in figure 4).

2.3 COING: A Practical Approach for Building a Generalization Space of Relational Descriptions

COING is an ascending method for building a GS: it relies upon the generalization of the object descriptions. In COING, objects are represented using graphs. In order to deal with the problem of matching graphs COING transforms the graph representation into an arc representation. In other words, each graph describing an object is transformed into a set of *independent* arcs. As such it can be considered as a transformation called propositionalization (Lavrac 01).

Instead of trying to match a graph G_1 with a graph G_2, COING only searches for a partial matching of an arc from G_1 with an arc from G_2 (Bournaud 97). This restriction has already been used in (Mineau 90). It has the advantage of limiting the complexity of the algorithm (c.f. section 2.3) because, as the arcs are oriented they fully match. However, this restricts the generalization language since the relations among arcs are not considered. In order to facilitate the representation and manipulation of the Generalization Space, it is built with one single root node.

This node clusters all the objects and it is the only node which may have an empty description. The root node have an empty description when the descriptions do not have a common part.

Example of a COING Generalization Space. In order to illustrate the COING approach, let us consider the three houses *h1*, *h2* and *h3* in figure 2 whose descriptions need to be organized. These houses are described by their windows which have two properties: a colour and a size. Figure 2 presents the GS built by COING for these houses.

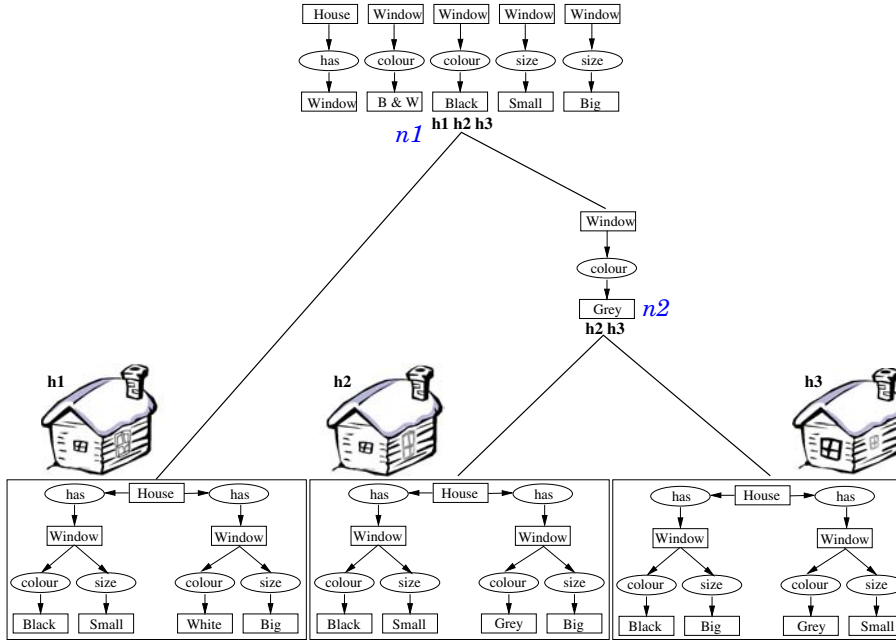


Fig. 2. Generalization Space built by COING from the three different houses.

Complexity of the COING Algorithm. Let E be the set of objects considered, $N = card(E)$ the number of objects and $n\_arcs(x)$ the number of arcs in the description of the object x. Let k_{max} be the maximum number of arcs in a graph describing an object and P_{max} the maximum depth in the generalization lattices (i.e. the size of the biggest path between the root and a leaf). The size complexity of COING depends on the generalization step. We define n_{ae} as the total number of arcs appearing in the object descriptions, $n_{ae} \leq k_{max} \times N$. During the generalization step, the maximum number of arcs that may be generated from an arc n_{ag} depends on the depth of the generalization lattice: it is in the order of $\theta(P^3_{max})$. Thus, the number of generated arcs n_a for all the arcs of the object descriptions is $n_a = n_{ae} + n_{ag}$. It is thus in $\theta(N \times k_{max} \times P^3_{max})$.

The time complexity of COING is linked to the generalization step. The usefulness of each generalized arc should be evaluated. In the worst case, the complexity is:

$$\begin{aligned}\theta(n_{ag} \times n_{ae}) &= \theta((N \times k_{max} \times P^3_{max}) \times (N \times k_{max})) \\ &\leq \theta((N \times k_{max} \times P^3_{max})^2)\end{aligned}$$

What makes this algorithm efficient is that both the number of generated nodes (n_a) and the time required to generate them are linear with the number of objects.

2.4 The Generalization Space Representation Language and Related Work

The reformulation used by COING of graphs into sets of independent arcs accounts for its quadratic complexity in the number of objects, but restricts the generalization language, i.e. the expressiveness of the descriptions of the GS nodes. This is because descriptions of the GS nodes consist of sets of independent arcs and not in sub-graphs[1]. Given a set of objects described as graphs, each node in the GS would ideally be represented by the graph that is the least general generalization of the graphs describing the objects it covers. Let us note this Generalization Space as GS_{max}. In fact, due to the complexity of the subsumption relation and the exponential growth of the length of the least general generalization, building GS_{max} directly using an exhaustive method is not practical.

This problem as been extensively discussed in the ILP community(Muggleton 94, Kietz 92). The syntactical restrictions on clauses (such as ij-determinacy) used to devise efficient ILP algorithms (Muggleton 94) are similar to the restrictions on graphs used to devise graph-based algorithms (Bournaud 97, Liquiere 98). Informally, most of the latter consider a generalization language that avoids matching graphs as much as possible. One of the sole systems that performs extensive (sub-)graph matching is SUBDUE (Cook 94) whose goal is to discover substructures in data using a fuzzy graph match. However it is concerned with repeated sub-graphs within one given object and does not address the problem of building a GS. Liquière and Sallantin Liquiere 98 proposed an algorithm that builds GS_{max} directly but, for tractability reasons, they require that the object descriptions be *locally injective graphs*. A locally injective graph (LIG) is formally defined as a labelled graph $G = (V, E, L)$, G is locally injective if for each vertex $v \epsilon V$ $\forall v_1, v_2 \in N^+(v), v_1 \neq v_2 \Rightarrow L(v_1) \neq L(v_2)$ and $\forall v_1, v_2 \in N^-(v), v_1 \neq v_2 \Rightarrow L(v_1) \neq L(v_2)$.

In other words, the matching complexity is such that efficient algorithm to build Generalization Spaces have ignored non-determinate relations. The solution we proposed in this paper is to build an initial GS using a propositional language and then to iteratively enrich the descriptions of its nodes. Intuitively our approach rely on the fact that in practice, although objects are describes by non-determinates relations, they are inherently not "hard" to generalize[2].

3 Using Abstract Relations to Cluster Relational Data

Let us call GS_1 the Generalization Space built by COING. This section explores other related abstractions and their use to iteratively enrich the Generalization Space GS_1.

[1] The three houses *h1*, *h2* and *h3* each have a small window and a black window; for *h1* and *h2* it is the same window, whereas for *h3* it is not. This difference does not appear in the classification built by COING (see figure 2) since it requires representing *relations* between two arcs.

[2] In the sense introduced by Giordana and Saitta (Giordana and Saitta, 00).

3.1 Abstract Relations

Underlying the approach proposed here is a view of particular sub-graphs as *abstract arcs*. The notion of abstract arc is based upon the notion of *abstract relation* defined as follows:

Definition 1. *(Abstract Relation): Given two concepts C_s and C_t in a graph, an abstract relation R_a corresponds to a relation between C_s and C_t denoted by the path between them. The level of an abstract relation corresponds to the number of vertex relations it contains.*

Definition 2. *(Abstract Arc): An abstract arc is an arc whose relation is abstract.*

Let us consider the following sub-graph made up of two connected arcs:

$$\textit{[House]}\rightarrow\textbf{(has)}\rightarrow\textbf{[Window]}\rightarrow\textbf{(size)}\ \rightarrow\textit{[Small]}$$

The triplet *(has)→[Window]→(size)*, which is in bold type, may be abstracted into a new relation $(Ra)_a$. The level of this abstract relation is 2, it contains two vertex relations. The associated abstract arc is

$$\textit{[House]}\rightarrow\ \textbf{(Ra)}_a\ \rightarrow\textit{[Small]}$$

In ILP terms, an abstract arc may be seen as a relational cliché(Silverstein 91) and an abstract relation to an invented predicate that correspond to this cliché.

3.2 Abstract Relations Based on Canonical Graphs

Given a graph language and depending on relations between concept vertices, there are numerous kinds of possible abstract relations (as they are of relational cliché in ILP). The number of reformulations of a graph into a set of abstract arcs typically increases with the number of abstract relations allowed, and there is a need for characterizing a set of abstract relations that will correspond to a particular generalization language. Our goal is to propose a set of abstract relations parameterized by a level.

Level l abstract relations correspond to sub-graphs consisting of l arcs. We have defined three canonical sub-graphs: *sequence*, *star* and *hole*. An example of a *sequence structure* used to build an abstract relation has been given in section 3.1; it is defined as a succession of arcs which are connected one to another thanks to a common concept. This concept is the target of the first arc and the source of the next one. The level of this abstract relation is 2. The three canonical sub-graphs *sequence*, *star* and *hole* were chosen for reasons of simplicity and genericity. We are currently formally demonstrating the completeness of these structures, which can be used to generate any graph. Other sub-graph structures could equally well be chosen, but they would have to respect this hypothesis in order to allow the method to really build any graph.

3.3 Propositionalizing Graphs into Arcs Using Abstract Relations

Given a set of abstract relations, a graph G_a describing an object may be reformulated into a set of abstract arcs. The main advantage of this reformulation is that, since the graph G_a is represented by a set of arcs, it may be performed using the propositional algorithm COING. The idea of the KIDS approach is to gradually reformulate the descriptions into a sequence of less and less abstract language, based on increasing levels of abstract relations. In the first step, KIDS manipulates level 1 structures: arcs. In the second step, KIDS manipulates level 2 structures composed of two arcs. In the i step KIDS manipulates level i structures composed of i arcs.

If the successive reformulations of the descriptions made by KIDS are easy to express when the descriptions are represented using a graph formalism, it is also possible to formulate them using a logical formalism. Figure 3 below presents the reformulation of the logical description of a house.

4 KIDS: An Algorithm to Cluster Relational Data Based on Abstract Relations

4.1 Principle of an Iterative Algorithm to Enrich the GS

Having seen the notions of abstract relations and abstract arcs, let us now examine the KIDS algorithm that incrementally builds a GS of relational descriptions

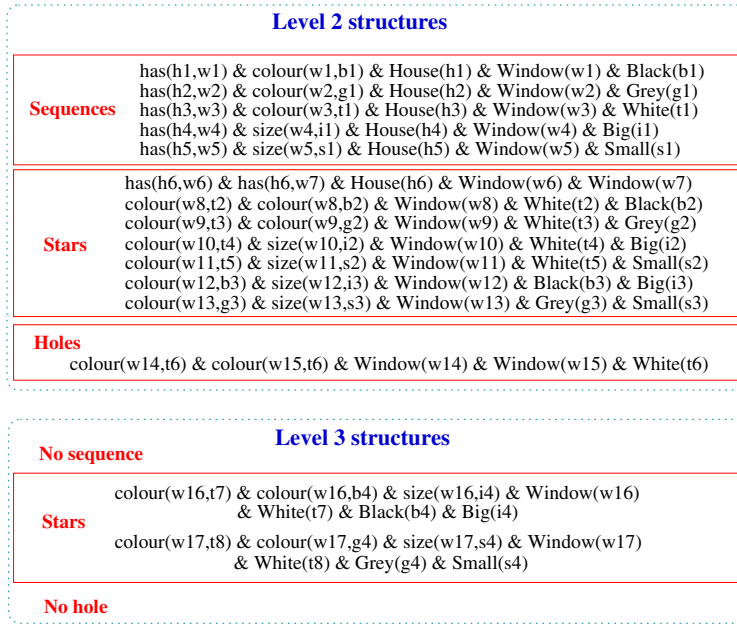


Fig. 3. Reformulation of a logical description using the canonical structures: star, sequence and hole.

by gradually increasing the level of the abstract relations used to reformulate the descriptions. In terms of generalization language, it means that KIDS considers a sequence of more and more abstract language. In the first step, the generalization language is an arc language; in the second step, the generalization language is a language of sub-graphs made up of two connected arcs (the descriptions are reformulated based upon level 2 structures); in the third step, it is a language of three connected arcs, and so on. In each step, object descriptions are reformulated into abstract arcs, and the reformulated descriptions are then organised by COING.

4.2 Enriching the Description of the GS Nodes

Whereas processing the reformulated descriptions using COING has a quadratic complexity (c.f. section 2.3), reformulating the descriptions requires sub-graphs to be matched and has a combinatorial explosion. It is obvious that the number of reformulations should be limited.

Let us state a property with respect to the Generalization Space:
If there exists a sub-graph Sgn which generalizes n object descriptions, then there is in the GS built by COING, called GS_1, a node whose coverage contains these n objects (and possibly others) and whose description contains all the arcs of the generalizing sub-graph Sgn.

In other words, if two descriptions do not have any common arcs (or arcs that may be generalized), it is useless to search for a common sub-graph between them. Indeed, a $(i+1)^{th}$ level structure is the aggregation of an i^{th} structure and one arc. At the i^{th} step, the KIDS algorithm does not explore all the nodes, but only those which may be enriched, i.e. the nodes whose descriptions potentially contain an i^{th} level structure. We define a candidate node as follows:

Definition 3. *(Candidate node): a GS node is a candidate node for KIDS at the i^{th} step if it has been modified at the $(i-1)^{th}$ step.*

4.3 KIDS Algorithm

The first step of KIDS consists in applying COING to the relational descriptions in order to build GS_1. Then, the enrichment algorithm of the GS is as follows:

1. Determine all the candidate nodes. In the second step of KIDS, it consists in all the nodes of GS_1.
2. For each object covered by a candidate node, determine its i^{th} level description: i connected arcs. This consists in reformulating the object descriptions using the three canonical structures of abstraction - sequence, star and hole - of i arcs.
3. Apply COING to the reformulated object descriptions, i.e. the set of abstract arcs describing the objects. The result of this application may be:
   - the modification of the description of an existing GS node. It is the case when a sub-graph is found in all the descriptions of the objects covered by this existing node,

- the addition of a new node to the GS. It happens when a sub-graph is found in a *sub-set* of the descriptions of the objects covered by an existing node. The new node is added as a son of the existing node,
- nothing, no common sub-structure of a higher level is found among the descriptions.

4. If KIDS modifies the GS at the i^{th} level, then repeat the method from 1) at the $(i+1)^{th}$ level.

Note that the GS node descriptions, formulated using abstract arcs, have to be reformulated in terms of sub-graphs. This consists in reformulating the descriptions using the abstract relations.

4.4 Example of a KIDS Generalization Space

Let us consider again the example of the houses presented in section 2.3 (figure 2) to illustrate the advantage of KIDS over COING. Figure 4 presents the enriched GS obtained by KIDS at level 2 using a type lattice that says that the colors *Black*, *White* and *Grey* are generalized in *W&B*. The information drawn in black is the result obtained by KIDS, that in grey is the result obtained by COING.

The use of abstraction allows us to discover common substructures in the object descriptions. At level 2, KIDS finds relational generalizations not found by COING. For example, COING found that all the houses have (at least) one window in common, KIDS found that all the houses have (at least) two windows in common. Likewise, KIDS allows us to know that all the windows have a colour (*W&B* or *Black*) and a size (*Small* or *Big*). Furthermore, a new class clustering *h1* and *h2* has been added to the Generalization Space: these two houses have a small black window in common and this window does not appear in the description of *h3* (even if *h3* has a small window and a black window but it is not the same window). This similarity is found by KIDS at level 2, because it is a special composition of two arcs. In this example, KIDS enriched the description of the GS's existing nodes and added a new node to GS_1, clustering *h1* and *h2*. Thus, from a GS built using a propositional language, KIDS has been able to give more precise information on the existing similarities among the objects thanks to an abstraction of sub-graphs.

For this example, it is useless to apply KIDS at level 3, since the descriptions of the houses *h1*, *h2* and *h3* do not contain level 3 stars, holes or sequences. Therefore, level 2 structures are enough to describe entirely the given houses *h1*, *h2* and *h3*.

4.5 Algorithmic Complexity and Maximum Level of Abstraction

From Matching for Generalization to Matching for Reformulation. The theoretical complexity of KIDS is in the worst case exponential since it requires relational descriptions to be matched. Though the complexity of matching for generalization is avoided by the use of abstract relations, the complexity of graph matching is not removed; it is instead moved to the reformulation of the

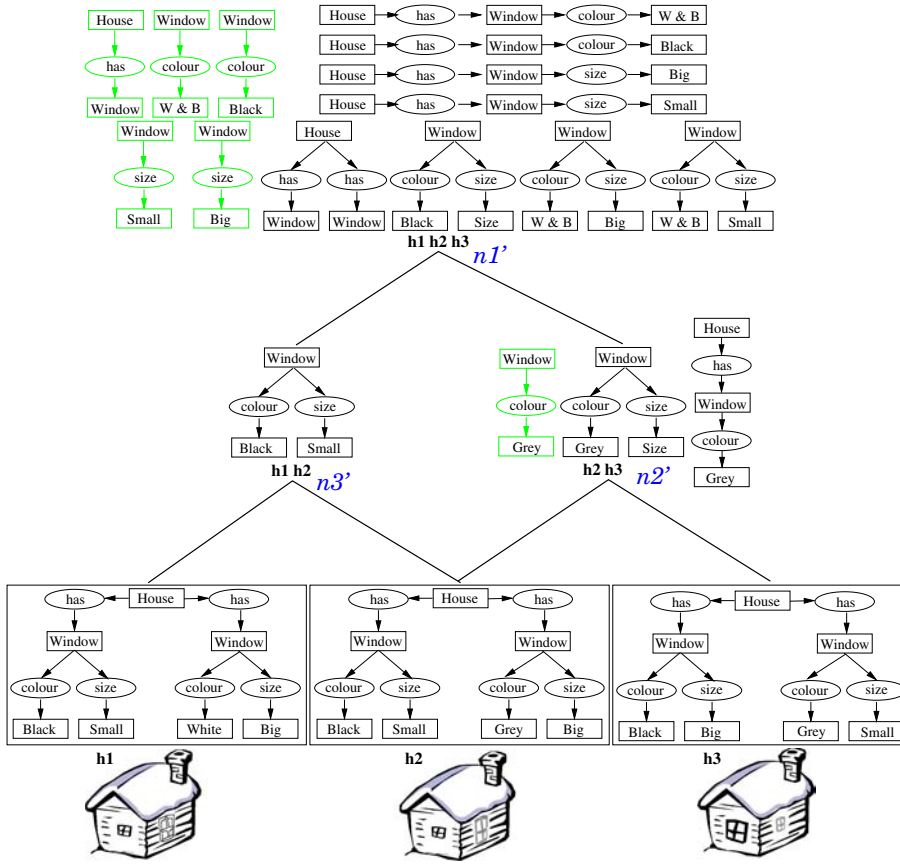


Fig. 4. Generalization Space enriched by KIDS.

descriptions. In fact, the more complex the abstract relations are (the higher the KIDS level), the more complex the reformulation is. Nevertheless, as explained previously, the specific structure of the GS and the iterative method of KIDS allow us to limit the number of nodes to be explored at each level, while exploring only those that may be enriched. In other words, it means that at each step of KIDS, for each abstract level, not all the descriptions of objects are reformulated but only useful ones. The main advantage of KIDS over a direct method to build a relational GS is that it *gradually* takes into account the relational part of the descriptions. KIDS may be stopped at any level, it gives a Generalization Space whose generalization language may be characterized in terms of level of abstraction. As such, KIDS may be seen as an "anytime algorithm".

In the current version, KIDS stops either when there are no more candidate nodes, or when it is not possible to describe the objects at the next level (there are no structures of (i+1) arcs in the object descriptions). However, we observed experimentally that KIDS does matching that is not useful. We have tried to characterize the uppermost level to which KIDS needs to be applied - w.r.t. to

the object description - in order to find all the relational similarities among the descriptions. We call this bound the *maximum level of abstraction*, and note this level l_{max}. We have formally defined this bound.

Experimentally, the time needed to apply KIDS at the next level may be evaluated from the time needed to perform the previous level. It is possible to approximate the time required for the next level and to stop KIDS if this time is too long. Experiments in section 5 show that, in our particular domain, the increase in time required between two successive levels is linear.

5 Experiments on Chinese Characters

The algorithms presented in this paper are developed in the CAC system[3]. CAC is a system for unsupervised learning of relational data and has been implemented in Java.

The domain considered for these experiments is related to the task of building classifications of Chinese characters for pedagogical purposes. Below is a brief description of the context of this work, the reader should refer to (Zucker & Ganascia, 98) for more information about this application.

5.1 Description of Chinese Characters as Relational Data

The database considered is a collection of 6780 Chinese characters. Each character is represented by a conceptual graph. Characters are described by the following characteristics: their initial and final pronunciation, the tone of this pronunciation, the various components (between 1 and 5) and their relative positions, and the so called key component. This database has been useful to experiment with the empirical complexity of KIDS, we do not discuss here the interestingness of the resulting Generalization Space for chinese experts (Zucker & Ganascia, 98).

5.2 Results and Discussion

We evaluated KIDS on several databases of characters composed of between 10 and 416 characters. Figure 5 shows the total time required for generating the GS for these databases using the COING and the KIDS algorithms. It appears in figure 5 that, in practice, the CPU time of the proposed algorithms is linear; in the worst case it is quadratic with the number of objects for COING (c.f. section 2.3). These results may be surprising because, since it manipulates sub-graphs, KIDS introduces a complexity factor. However, the combinatorial explosion due to the generalization of sub-graphs is limited since the higher the level (i.e. the more complex the graphs to be generalized), the less fewer sub-graph matching needs to be performed. The level of abstraction introduces a multiplicative factor. The linear growth means that, on the average, the time needed to move to the next level is very close to constant. Figure 6 below illustrates this result.

[3] The CAC system is registered to the University of Paris 6 under the number ABO100000132034544.

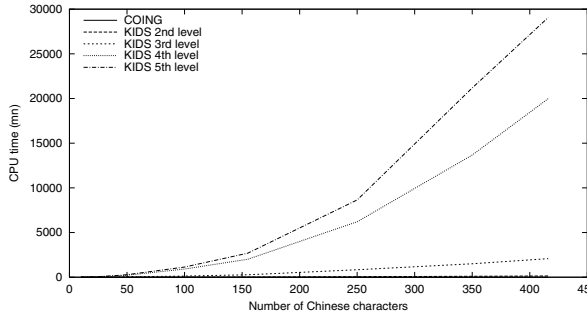


Fig. 5. Average execution time of COING and KIDS on Chinese characters benchmark databases.

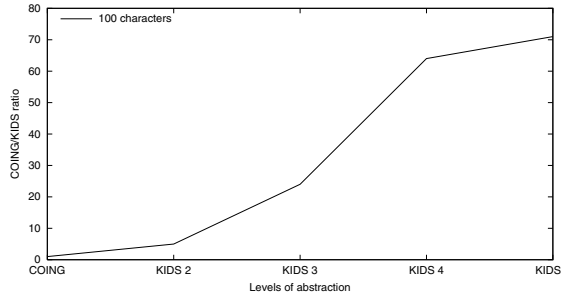


Fig. 6. Empirical evolution of the multiplicative factor as a function of the abstraction levels.

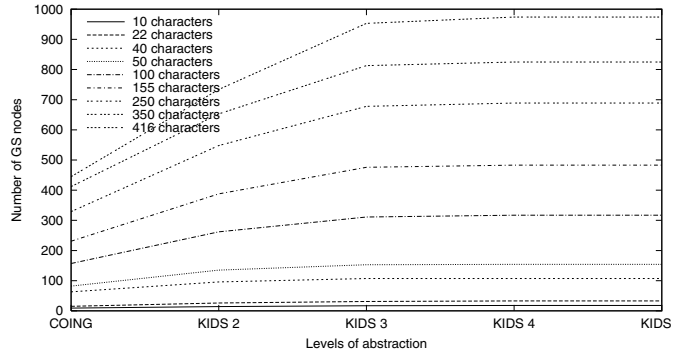


Fig. 7. Evolution of the number of GS nodes.

During these experiments, we also evaluated the evolution of the number of GS nodes as a function of the algorithms used. For COING, in the worst case, this number is in $\theta(N)$. Figure 7 summarizes these results.

This graph shows that the number of nodes in the GS grows until a specific level - level 1 for the small bases and level 2 for the large ones - then it becomes

constant. This may be explained by the fact that from a specific level, KIDS does not allow the creation of new classes, but only enriches the already existing ones with more complex descriptions.

6 Conclusion

Organizing relational data has many applications in the field of data mining, knowledge indexation or systematic but may also be used to extract conceptual hierarchies with particular properties. The problem of designing efficient algorithms for this purpose is hard because of the known graph-matching complexity. This paper has proposed an approach to introduce the complexity of the relations incrementally. It is based on abstraction and consists in reformulating the description using a sequence of languages that are less and less abstract. The reformulated descriptions are processed using a propositional learner. The KIDS algorithm builds an initial GS using a propositional language and then iteratively enriches the descriptions of its nodes.

Our experiments suggest that the proposed method provides an organization of relational descriptions while keeping a linear complexity in practice with the number of objects. This result is due to the fact that the more complex the structures, the fewer descriptions need to be performed. Our work supports the idea that iterative abstraction may be an appropriate approach to deal with the traditional representation trade-off. By increasing the expressiveness of the language, the solution is refined at each step and results from the previous step are used to reduce the complexity of the current step. Moreover, preprocessing the data makes it possible to evaluate the maximum level of effectiveness of KIDS, called the l_{max}. Even if the data are described using a relational description, their structure may not be relational and using a propositional learner is enough to discover their similarities.

A perspective of this work is to extend this method for a more efficient processing of numerical data. Currently, the numerical information contained in descriptions is processed as symbols; the implicit order existing between numbers is not taken into account. One of our application domain is related to the organization of knowledge of a group of genes that play a key role in the regulation of energy balance (Clément 00). The detection of such a group is, by itself, a numerical clustering task where genes of similar expressions are clustered (Ben-Dor 99). Each gene is described by symbolic and numerical data.

Another direction of research is to characterize logically the generalized language used at each step of KIDS to enrich the GS. We are currently clarifying the link between the successive abstract languages used by KIDS for reformulation and the notion of "k-locality" defined by Cohen (Cohen 94).

Acknowledgments

The authors wish to thank ILPnet2 which allows us to attend to the conference.

References

Ben-Dor A., Shamir P. & Yakhini Z.: Clustering Gene Expression Patterns. *Journal of Computational Biology*, 6(3/4). (1999) 281-297.

Bournaud I., Ganascia J.-G.: Accounting for Domain Knowledge in the Construction of a Generalization Space, *ICCS'97*, Lectures Notes in AI, 1257, Springer-Verlag (1997) 446-459.

Carpineto C., Romano G.: GALOIS: An order-theoretic approach to conceptual clustering, *Tenth International Conference on Machine Learning* (ICML) (1993).

Cheeseman P., Kelly J., Self M., Stutz J., Taylor W., & Freeman D.: AutoClass: A Bayesian Classification System. *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, MI. June 12-14 1988. Morgan Kaufmann Publishers, San Francisco. (1988) 54-64.

Chein M., Mugnier M.L.: Conceptual Graphs: Fundamental Notions, *Revue d'Intelligence Artificielle*, 6(4) (1992) 365-406.

Clément K.: Monogenic forms of obesity: from mouse to human. *Annal of Endocrinology*, 61(1). (2000) 39-49.

Cohen W.: Pac-learning Nondeterminate Clauses. *Twelth National Conference on Artificial Intelligence*. (1994) 676-681.

Cook D. J., Holder L. B.: Substructure discovery using minimum description length and background knowledge, *Journal of Artificial Intelligence Research* 1 (1994) 231-255.

Dehaspe L., Toivonen H.: *Frequent query discovery: a unifying ILP approach to association rules mining*. Technical Report CW-258, Department of Computer Science, Katholieke Universiteit Leuven, March 1998.

De Raedt, L. (1998). Attribute-Value Learning versus Inductive Logic Programming: The missing links. European Conference on Machine Learning (ECML), Springer-Verlag.

De Raedt, L. and H. Blockeel (2001). Using Logical Decision Trees for Clustering. IJCAI 2001.

Diday E.: La méthode des nuées dynamiques. *Revue de Statistique Appliquées*, XIX(1). (1971) 19-34.

Fisher D.: Approaches to conceptual clustering, *Ninth International Joint Conference on Artificial Intelligence* (IJCAI), Los Angeles, CA, Morgan Kaufmann (1985).

Fisher D.: Knowledge Acquisition Via Incremental Conceptual Clustering, *Machine Learning: An Artificial Intelligence Approach*, R. Michalski, J. Carbonell and T. Mitchell, San Mateo, CA, Morgan Kaufmann, II (1987) 139-172.

Fisher D.: Iterative Optimization and Simplification of Hierarchical Clusterings. *Journal of Artificial Intelligence Research* **4** (1996) 147-179.

Flach P. and Lachiche N.: Confirmation-Guided Discovery of First-Order Rules with Tertius. *Machine Learning, Special issue on unsupervised learning*, Vol. 42, Number 1/2, Kluwer, (2001) 61-95.

Gennari J. H., Langley P., Fisher D.: Models of incremental concept formation, *Artificial Intelligence* 40-1(3) (1989) 11-61.

Giordana, A. and L. Saitta (2000). Phase Transitions in Relational Learning. *Machine Learning Journal* 41(2): 217-.

Haussler D.: Learning Conjunctive Concepts in Structural Domains, *Machine Learning* (4), (1989) 7-40.

Ketterlin A., Gancarski P., Korczak J.J.: Conceptual clustering in Structured databases: a Practical Approach, *Proceedings of the Knowledge Discovery in Databases*, KDD'95, AAAI Press (1995).

Kietz, J.-U. and S. Wrobel (1992). Controlling the Complexity of Learning in Logic through Syntactic and Task-Oriented Models. Inductive Logic Programming. S. Muggleton. London, Harcourt Brace Jovanovich: 335-359.

Lavrac, N. and P. A. Flach (2001). An extended transformation approach to inductive logic programming. ACM Transactions on Computational Logic 2(8): 458-494.

Liquiére M., Sallantin J.: Structural Machine Learning with Galois Lattice and Graphs, *Fifteen International Conference on Machine Learning*, (ICML), (1998).

Michalski R. S., Stepp R. E.: An application of AI techniques to structuring objects into an optimal conceptual hierarchy, *Seventh International Joint Conference on Artificial Intelligence*, (IJCAI) (1981).

Mineau G., Gecsei J., Godin R.: Structuring knowledge bases using Automatic Learning, *Sixth International Conference on Data Engineering*, Los Angeles, USA (1990).

Muggleton, S., Raedt L. D.: Inductive Logic Programming: Theory and Methods, *Journal of Logic Programming* 19(20) (1994) 629-679.

Sowa J. F.: *Conceptual Structures: Information Processing in Mind and Machine*, Addisson-Wesley Publishing Company (1984).

Silverstein, G. and M. J. Pazzani (1991). Relational clichés: Constraining constructive induction during relational learning. 8th IWML, Morgan Kaufmann.

Wille, R.: Restructuring Lattice Theory. *Symposium of Ordered Sets*, I.Rival (eds). (1982) 445-470.

Wilson D.R. & Martinez T.R.: Reduction Techniques for Instance-Based Learning Algorithms. *Machine Learning*, 38(3). (2000) 257-268.

Zucker J.-D., Ganascia J.-G. Changes of Representation for Efficient Learning in Structural Domains. *International Conference in Machine Learning (ICML'96)*, Morgan Kaufmann (1996).

Zucker, J.-D., J.-G. Ganascia, et I. Bournaud (1998). Relational Knowledge Discovery in a Chinese Characters Database. *Applied Artificial Intelligence* 12(5): 455-488.

Efficient and Effective Induction of First Order Decision Lists

Mary Elaine Califf

Department of Applied Computer Science, Campus Box 5150, Illinois State University
Normal, IL 61790 USA
mecalif@ilstu.edu

Abstract. We present BUFOIDL, a new bottom-up algorithm for learning first order decision lists. Although first order decision lists have potential as a representation for learning concepts that include exceptions, such as language constructs, previous systems suffered from limitations that we seek to overcome in BUFOIDL. We present experiments comparing BUFOIDL to previous work in the area, demonstrating the system's potential.

1 Introduction

In machine learning, there are a variety of metrics that can be applied to algorithms. An important measure of any learning algorithm is, of course, predictive accuracy. However, the degree to which the representation used fits the problem and the comprehensibility of the resulting set of rules can also have a significant impact on the usefulness of a system.

Our primary area of interest is in applying machine learning algorithms to language problems. Here, much of the successful work has been statistical in nature. However, statistical methods are limited in their expressiveness and produce rules that are not easy for humans to understand. Therefore, we believe that work in language learning that uses more expressive representations that are more amenable to human understanding is desirable.

One rule representation that seems to be a very good fit for language learning is that of decision lists. Decision lists are particularly good for representing concepts with general rules that have exceptions. In previous work, we showed that *first-order decision lists* are a highly effective representation for learning a morphological concept (specifically generating the past tense form of a verb given the base form) [8].

However, the FOIDL system has one major flaw: efficiency. FOIDL is very effective when run on a fairly small example set and produces a very understandable set of rules. However, it is a top-down algorithm that quickly blows up in the face of large numbers of examples and a large search space of constants, as will be demonstrated below. This property limits the practical usefulness of the algorithm, since language learning generally involves a large search space and a large number of examples.

This issue has been previously addressed by Manandhar, Džeroski, and Erjavec [7], who recognized the potential of FOIDL's representation and attempted to address the

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 17–31, 2003.

efficiency problems of the algorithm in their CLOG system. CLOG is a very fast algorithm. However, it has its own set of flaws, particularly in terms of generality and accuracy, as will be discussed further below.

We have developed BUFOIDL, a system that addresses the issues of effective and efficient induction of first order decision lists, providing accuracy comparable to that of FOIDL and time complexity that (although not rivaling CLOG's speed) is far more acceptable than that of FOIDL.

The remainder of this paper is organized as follows: Section 2 presents background material on first order decision lists, FOIDL, and CLOG. Section 3 presents the BUFOIDL algorithm. Section 4 presents experimental results comparing the three systems. Section 5 briefly discusses other relevant work. The final section concludes and presents some directions for future work.

2 Background

In this section, we explain what we mean by first-order decision lists and briefly describe the two systems that inspired our work: FOIDL and CLOG.

2.1 First-Order Decision Lists

First-order decision lists are ordered sets of clauses, each of which ends in a cut. Thus, as the clauses are tried in order, only the answer produced by the first matching clause is obtained. This is basically a first-order extension of prepositional decision lists [13], where each rule consists of a set of tests and a category label, and an example is assigned to the category label for the first rule such that the example meets the tests. Initial work in this area learned rules in the order in which they are applied [13,6], but Webb and Brki•[14] pointed out the advantages of learning the rules in reverse order, since more general rules tend to be learned first given most preference functions. FOIDL takes this approach, and we follow it as well.

2.2 The FOIDL Algorithm

FOIDL is a top-down inductive logic programming algorithm based largely on FOIL [11]. To FOIL, FOIDL adds three primary features:

- the ability to handle intensionally defined background predicates,
- the ability to handle implicit negative using a concept of output completeness,
- and the ability to learn first-order decision lists.

The first of these is fairly straightforward, but is one of the causes of time spent by the algorithm. FOIDL must actually execute each rule formed for each example because of the intensional background.

The second distinguishing feature of FOIDL is its use of an output completeness assumption to handle implicit negative examples. In order to make use of this assumption, the predicate to be learned must have a *mode* associated with it, indicating which arguments are input(s) and which are output(s). Then the assumption may be made that for any unique input pattern for which a positive example is provided, all

other positive examples with that input pattern are also in the training set. Clearly, this is the case for any predicate that represents a function with a unique output for each input.

Once we have this assumption, we can quantify the number of implicit negative examples covered by a clause. For each example, we produce an *output query* that specifies the inputs. If the clause applied to the output query produces a ground answer that does not match a correct output for the input, it covers a single negative for that example. If it produces a non-ground answer, FOIDL estimates the number of examples covered based on a parameter indicating the size of the universe from which an answer might be taken.

The output completeness assumption is not specific to the induction of first-order decision lists and has been used in learning relations that are not functions [4,5]. Note that it is less restrictive than the closed world assumption in the sense that the system must be guaranteed only all correct outputs for each input actually present in a given data set. For further discussion of the advantages of the output completeness assumption, see [4].

The third distinguishing feature of FOIDL is its ability to learn first-order decision lists. It does this by first learning a clause that covers as many positive examples as possible. Note that this clause may produce incorrect answers for examples that have not yet been covered. For the case of producing the past tense in English, this might be a rule such as:

```
past(A,B) :- split(B,A,[e,d]).
```

where *split* is a predicate whose second and third arguments are non-empty lists that produce the first argument when appended together.

In developing subsequent rules, clauses are constrained to not cover previously covered examples. So the next rule to be learned might be:

```
past(A,B):- split(B,A,[d],
       split(A,C,[e]).
```

where the literal that constrains *A* to end in *e* is required to keep this special case rule from firing for examples that should be covered by the default rule.

The basic algorithm for learning a clause in FOIDL is explained in Figure 1.

This is complicated slightly be the idea of exceptions to exceptions. For example, consider the case of a verb ending in *y* in English. To produce the past tense, you typically remove the *y* and add *ied*. However, to learn this rule requires covering some examples that end in *y* but still add *ed* to produce the past tense (consider *delay*). To accommodate this, FOIDL will "uncover" previously covered positive examples (adding them to *positives-to-cover*) if doing so allows the system to learn a rule that covers a sufficient number of positive examples.

FOIDL seems to be very effective at learning concepts that fit the first-order decision list representation. However, it does have some drawbacks. It requires that all constants be explicitly specified, since it works exclusively in a top-down fashion. For the past tense problem, this requires the generation of all possible prefixes and suffixes that appear in multiple examples, and it requires searching through that space. The primary issue with FOIDL is simply that it cannot be applied to too many examples, particularly if the search space of constants and/or background predicates is large.

```
Initialize C to R(V1,V2,…Vk) where R is the target
        predicate with arity k
Initialize T to contain all examples in positives-to-
        cover and output queries for all positive
        examples
While T contains output queries
   Find the best literal L to add to the clause
   Let T' be the subset of positive examples in T whose
           output query still produces a first answer
           that unifies with the correct answer, plus
           the output queries in T that either
      1)Produce a non-ground first answer that unifies
                with the correct answer, or
      2)Produce an incorrect answer but produce a
                correct answer using a previously
                learned clause
Replace T by T'
```

Fig. 1. FOIDL algorithm for learning a clause

2.3 CLOG

The CLOG system was developed because attempts to apply FOIDL to other problems showed its limitations in regard to processing speed [7]. CLOG shares a number of characteristics with FOIDL, but takes a different approach to learning individual clauses.

Like FOIDL, CLOG uses the concept of output completeness to allow for implicit negatives and it produces first-order decision lists in the same order as FOIDL, prepending each new clause to the decision list.

However, the way in which CLOG learns rules is quite different. In the following discussion, PTC represents positive examples not covered by the decision list, and CPE represents positive examples already covered by the decision list. Until all examples are in CPE, CLOG selects an arbitrary example and creates all of the possible clauses that cover that example (using a user-defined predicate that accepts an example and returns the appropriate clauses for the example). For each of those clauses, it counts the number of examples in PTC that the clause covers positively and negatively and then the number of examples in CPE that are positively or negatively covered by the clause. Given those four counts, CLOG calls a user-defined gain function to determine which of the clauses is best. Once a clause has been selected, examples positively covered by that clause are removed from PTC and added to CPE and examples negatively covered by the clause are added to PTC and removed from CPE.

This approach has some advantages over that of FOIDL. First, it is very fast, as will be shown in the discussion of experimental results. Second, it does not require the search through the space of constants, since those are created in the development of the possible covering clauses. Third, the management of exceptions to exceptions is integral to the algorithm.

However, the approach also has its own set of drawbacks. It requires that the user of the system specify a predicate to generate the possible clauses covering a given example. In the case of morphological learning, the domain to which CLOG has been applied, this is relatively straightforward; however, it is easy to imagine cases where it would not be easy to do. The system is also being given a considerable amount of knowledge about what a clause is "supposed" to look like, so that needs to be taken into consideration when doing comparisons.

The second major drawback of the system is that the arbitrary choice of an example for building a rule means that rules are likely to not be constructed from most general to least general. An examination of the rules produced by CLOG quickly shows that it often fails to learn a general "default" rule. Manandhar, Dzeroski, and Erjavec [7] point this out as a positive feature in comparison to FOIDL, since the system is more likely to fail to produce any answer than to produce an incorrect answer. However, we would argue that a wrong answer may be more desirable than no answer at all. When trying to generate the past tense form of "steal", we believe that it is better to produce "stealed" than to produce nothing, since the incorrect answer is, in fact, comprehensible, even though it is wrong. Certainly, there are cases where it may be desirable to fail instead, but first-order decision lists seem to be most applicable to those cases where a default answer is likely to be appropriate. We will also show that CLOG's accuracy is not consistently competitive with FOIDL's.

3 BUFOIDL

The BUFOIDL (Bottom-Up First Order Induction of Decision Lists) algorithm was inspired by a desire to overcome the limitations of FOIDL without trading off accuracy and flexibility. Since the primary cause of FOIDL's long learning times stems from its top-down search through a large space of constants and possible literals, we decided to take a bottom-up approach to the problem.

Many characteristics of BUFOIDL come directly from FOIDL. BUFOIDL handles intensionally defined background predicates in much the same way as FOIDL. It uses the output completeness assumption to handle implicit negative examples just as FOIDL and CLOG do. It constructs the decision list in the same way as FOIDL, first learning a most general clause, then learning more specialized "exceptions" to the clauses below. BUFOIDL also uncovers positive examples under the same circumstances as FOIDL.

Thus, the algorithm for the outer loop is very similar to FOIDL's outer loop. BUFOIDL, however, takes a different approach to the construction of individual clauses, inspired by previous bottom-up approaches, particularly from GOLEM [9] and PROGOL [10].

3.1 Clause Construction

BUFOIDL constructs a group of potential clauses and selects from that group the best next. The clause construction algorithm, which is very close to GOLEM, much as FOIDL follows FOIL, is shown in Figure 2.

Note that old-clauses parameter refers to clauses other than the one selected on previous iterations. It does not include clauses from the current definition.

```
Get-Generalizations(PTC,PCE,old-clauses)
 Initialize example-pool to PTC
 Set prev-clauses to empty
 For each cur-clause in old-clauses
   Evaluate cur-clause
   If cur-clause covers more positives than negatives
      Remove covered positives from example-pool
   Else old-clauses = old-clauses - clause
 Repeat
   Select
     k pairs of examples from example-pool
     k pairs of one example from example-pool and one
         clause from prev-clauses
     k pairs of clauses from prev-clauses
   Generate the most-specific clause covering each
         selected example
   For each pair
      cur-clause = the LGG of the pair
      Evaluate cur-clause
      If cur-clause covers more positives than
            negatives
         Add cur-clause to prev-clauses
         Remove covered positives from example-pool
         Move parent(s) of cur-clause to old-clauses
 Until no new clauses are found
 Return old-clauses + prev-clauses
```

Fig. 2. BUFOIDL algorithm for creating a set of generalizations from which to select a new clause.

Several aspects of this algorithm require explanation. First, the concept of a "most-specific clause" is a difficult issue, and one of the problems that all bottom-up approaches to ILP must address in some way. We allow intensionally defined background predicates, so we cannot use RLGGs as GOLEM does [9]. We chose not to require extensive information about the syntactic form of the learned clause as systems such as CLOG and PROGOL do [7,10]. Instead, we take the approach of using only type and mode information for each of the predicates, and applying the background predicates to the initial example in all possible ways, collecting the resulting literals and using the conjunction of all of those literals as the "most-specific clause". For some problems, this process might need to be repeated multiple times, and could lead to significant increase in complexity of the LGG process, but BUFOIDL attempts to control this complexity in two ways. A parameter is provided to limit the maximum times the technique is applied, and BUFOIDL attempts to learn clauses using a single level, using multiple applications of the technique only when the system fails to learn a new clause (on the principle that the system is designed to learn more general clauses first). As an example of the construction of a most-specific clause, consider Figure 3, taken from the English past tense task.

```
Example: past([k,6,m],[k,e,m]).
Type: past(word,word)
Mode: past(+,-)

Background: split(X,Y,Z)
Type: split(word, prefix, suffix)
Mode: split(+,-,-) or split(-,+,+)

Most-specific clause:
    past([k,6,m],[k,e,m]) :-
        split([k,e,m],[k],[e,m]),
        split([k,e,m],[k,e],[m]),
        split([k,6,m],[k],[6,m]),
        split([k,6,m],[k,6],[m]).
```

Fig. 3. Example of most-specific clause generation in BUFOIDL. split(X,Y,Z) is equivalent to append(Y,Z,X) with non-empty Y and Z

For some problems, this collection of literals needs to be done multiple times, so BUFOIDL has a *depth* parameter that determines how deep this search for literals is permitted to go. The system then performs an iterative deepening search, increasing the depth of the literal collection process when the algorithm fails to produce an acceptable new clause.

For the past tense problem that we focus on here, a depth of 1 is sufficient, and, in fact, the types and modes used by FOIDL and BUFOIDL for the problem, as well as the clause construction predicate used with CLOG, all constrain the learned rules to have a depth of 1. This actually may limit the ability of the representation to appropriately generalize exceptions such as *drink-drank*, but it is not a problem for the task in general.

Clauses are evaluated using output queries as described in Section 2.2. Because clause creation is bottom-up, we need not be concerned with estimating the number of negatives covered by non-ground responses. We simply eliminate any clause that produces non-ground answers as overly general. Note that negative examples come in two forms: BUFOIDL allows for explicit negative examples, which are treated as negatives when covered in the usual way, but we also consider as negative any incorrect answer to an output query if and only if some previously learned clause produces a correct answer to that query. Wrong answers for positive examples that have not yet been correctly covered are ignored, since we assume that they are exceptions to the current rule which will be handled by a rule that will be learned later (thus appearing earlier in the decision list).

Note that clauses covering examples negatively are kept for consideration and generalization if they cover more positives than negatives (i.e. produce more correct than incorrect ground responses). This is to allow for the possibility of exceptions to exceptions as discussed in Section 2.2 and below.

Another important aspect of BUFOIDL's clause construction is the process of search for a good generalization. Rather than randomly selecting one or several pairs and then generalizing each of those as much as possible without over-generalizing, BUFOIDL attempts a wider search, randomly selecting pairs of examples and rules

```
PTC = positive-examples
old-clauses = {}
CPE = {}
dec-list = empty
while PTC not empty and not done
   gen-list = Get-Generalizations(PTC,CPE,old-clauses)
   if gen-list is empty
      done = true
   else if some gen in gen-list covers no negatives
      best-gen = the generalization covering no
           negatives and the most positives
      Add best-gen to beginning of dec-list
      newCPE = the positives covered by best-gen
      PTC = PTC - newCPE
      CPE = CPE + newCPE
      old-clauses = gen-list - best-gen
   else
      best-gen = the generalization with the greatest
           difference between # of positives covered
           and # of negatives covered
      newPTC = the positive examples corresponding to
           the negatives covered by best-gen
      PTC = PTC + newPTC
      CPE = CPE - newPTC
      old-clauses = gen-list
```

Fig. 4. Algorithm for building the decision list in BUFOIDL.

repeatedly in a single clause construction phase and generalizing each pair as little as possible. The reason for this approach is that we would like to select the single most general clause possible at each step. This is not as important for an unordered set of rules such as GOLEM constructs. Of course, our approach cannot guarantee finding the best clause, but no heuristic search method can. Like GOLEM, BUFOIDL prunes the learned clause, dropping unnecessary literals.

Finally, note that *k*, the number of pairs to be generalized at each iteration, is a parameter to the algorithm. It should be set with some attention to the expected number of rules in the decision list to help ensure that at least one acceptable generalization will be found. Too many pairs should not harm the quality of the decision list learned, but will increase learning time, since the algorithm's execution time is highly dependent on the number of pairs selected at each pass.

3.2 Building the Decision List

As stated previously, BUFOIDL builds its decision list from the bottom up, placing the first, most general, rule learned at the end and prepending newly learned clauses to the list. However, a few details of the algorithm merit further explanation. The basic algorithm appears in Figure 4.

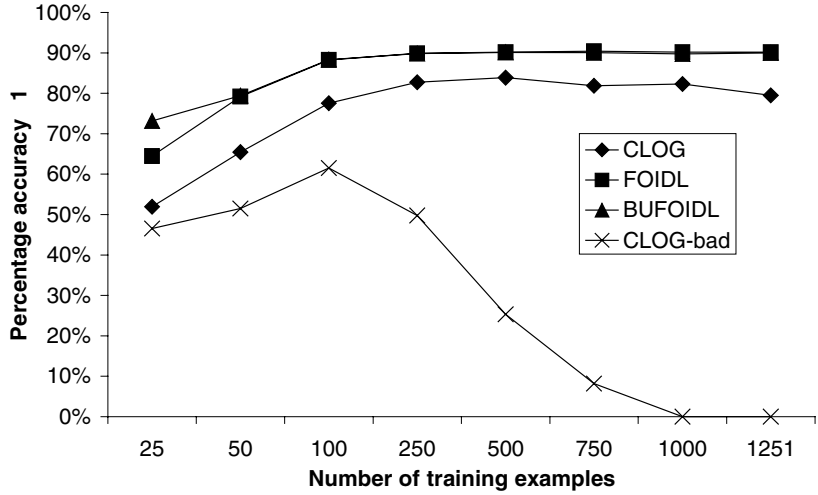


Fig. 5. Accuracy on full past tense task.

In order to make the search for clauses more efficient, BUFOIDL saves all of the acceptable clauses from previous iterations and evaluates them at the beginning of the search for a new clause, removing examples they cover from the pool to be generalized from if the clause is acceptable. While this is unlikely to helpful in all cases, as after the creation of the default clause, it may be very helpful later, when two clauses to be learned may not interact.

For example, in handling English plurals, we may learn a default clause that adds *s* to a word. We then may create two clauses, one that adds *es* if the word ends in *s* and one that adds *es* if the word ends in *z*. Both are clauses that we want in the decision list, and they do not interact with each other. BUFOIDL will be able add the clause that covers more examples and save the other to add in the next iteration of the loop without having to reconstruct it. Given BUFOIDL's broad search, this is important to the overall efficiency of the algorithm.

4 Experimental Evaluation

To evaluate BUFOIDL, we ran experiments using the English past tense task for which FOIDL was initially developed. This data set is one that all three systems can easily be applied to; it is fairly well known; and it is large enough to allow us to determine whether BUFOIDL actually overcomes the performance problems that limit FOIDL.

4.1 Experimental Design

The data used consists of 1390 verbs paired with their past tense forms in UNIBET phonemic encoding. The task is to generate the past tense form of the verb given the

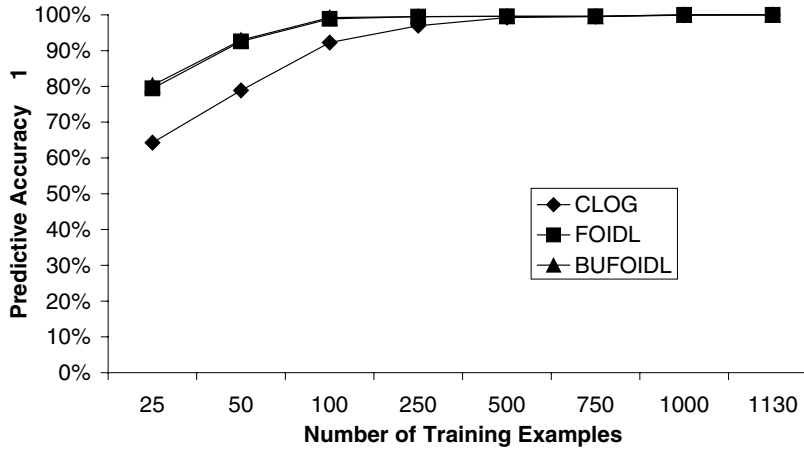


Fig. 6. Performance on past tense task using only regular verbs.

base form. We actually performed two experiments: one using the full set of verbs and a second that used only the regular verbs. The second task is much simpler than the first, and allows for 100% accuracy in theory (as the full task does not).

All three systems used the background predicate *split*/3 described earlier in the paper. For CLOG, we used an intermediate predicate called *mate* that simplifies the construction of the possible generalizations of an example. All of the user-defined portions of CLOG are used as provided for the tasks of generating English plurals. This is an example that comes with the system, and is highly similar in nature to the past tense task. The primary parameter for BUFOIDL is the number of pairs to select.

For the regular task, the number of pairs was set to 10, and we used 25 pairs when irregular verbs were included, since there is a much larger number of clauses to be learned in this case. In these experiments, we used a 10-fold cross-validation of the data, and also ran learning curves. Statistical significance was determined using 2-tailed paired t-tests.

4.2 Experimental Results

Figure 5 shows the results of running the three systems on the full past tense task. BUFOIDL and FOIDL perform well and very similarly on the task. With very few examples, BUFOIDL actually outperforms FOIDL, though FOIDL does slightly better than BUFOIDL with large numbers of examples. The differences between these two systems are attributable to the different search mechanisms. Like its predecessor FOIL, FOIDL uses a hill-climbing approach that is typically effective but can fail. BUFOIDL uses a broader search through the space, but it does have a random element. It seems that each approach can outperform the other at times, but there are no statistically significant differences between the two systems.

It is important to note, regarding CLOG's performance on the past tense task, that running these experiments highlighted one of the potential problems with CLOG's approach. As explained earlier, CLOG requires that the user of the system supply predicates to construct the set of clauses that are possible generalizations of an exam-

Table 1. Learning times in seconds for the full phonetic past tense task using different numbers of examples

	FOIDL	CLOG	BUFOIDL
25	1.003	0.342	178.431
50	4.541	1.076	199.121
100	25.379	3.343	204.16
250	407.225	11.487	179.859
500	3956.708	30.505	201.558
750	16,102.84	59.296	208.714
1000	42,237.62	92.733	188.925
1251	92,623.57	132.612	241.837

ple, to specify whether a clause is a generalization of another clause, and to specify a gain function. In order to run the past tense task, we used predicates for these purposes that the system developers supply for the task of producing English plurals, since that task has many similarities to the past tense task. However, the supplied predicates assume that the original word and the modified word (in the original task, the singular and plural forms; in our case, the base form and the past tense form of the verb) share a common prefix. In general, this is the case, but our data set includes the pair *go-went* and *eat-ate*, for which the assumption does not hold true. The presence of either of these verb pairs in a training set caused CLOG to fail without producing a decision list that could be tested. This situation let to the results label CLOG-bad. Therefore, we re-ran the experiments on slightly modified training sets from which those two verb pairs have been removed. The performance of FOIDL and BUFOIDL was identical, but CLOG's performance improves. Note that the removal of these verbs, rather than reducing potential accuracy, actually makes the task of the learner slightly easier, since both pairs function purely as noise in learning the decision list.

Clearly, this task demonstrates that CLOG is not always competitive with FOIDL and BUFOIDL in terms of predictive accuracy. Its approach seems to suffer greatly from the large number of exceptions, and accuracy is actually lower with larger numbers of examples. All of the differences between CLOG and BUFOIDL and those between CLOG and FOIDL with more than 25 training examples are statistically significant at the 0.01 level or better.

Figure 6 shows the result of running the systems on just the regular verbs. Here the systems are more comparable. FOIDL and BUFOIDL again perform very similarly, while CLOG is significantly lower with 250 examples or fewer, but eventually catches up to the other two systems.

So far we have shown that BUFOIDL has accuracy comparable to FOIDL's, but this is not sufficient to motivate the development of a new algorithm. The other issue is learning time. Table 1 shows the learning times for the complete phonetic past tense task, and Table 2 shows the learning times for the task involving regular verbs only.

Table 2. Learning times in seconds for the regular past tense task using different numbers of examples

	FOIDL	CLOG	BUFOIDL
25	0.771	0.298	86.802
50	3.347	0.941	87.525
100	18.636	2.22	93.101
250	230.096	7.777	109.334
500	1716.971	15.422	121.364
750	5586.535	24.072	101.839
1000	15,930.2	32.318	116.58
1125	17,951.2	36.391	113.085

These numbers clearly demonstrate the issue of learning time in FOIDL. While the system learns quickly from a small number of examples, the learning time increases rapidly, quickly becoming unreasonable. On the full phonetic past tense task, FOIDL averages over 25 hours of CPU time to learn from 1251 examples. This problem is exacerbated by a related increase in memory use that makes it difficult to run larger problems at all.

The learning times also show that CLOG's designers achieved their goal of creating decision lists in far less time than FOIDL requires. The difference is dramatic. It is interesting to note that the increasing number of exceptional cases seems to have a strong impact on both CLOG's and FOIDL's learning times. However, it is very clear that CLOG can handle many more examples than FOIDL.

Unlike FOIDL and CLOG, BUFOIDL's time depends less on the size of the training set than on the number of pairs selected. Because of this, the system takes quite a bit longer than the others on small example sets. In the full phonetic past tense task, the time required by BUFOIDL does not even show a clear trend toward increasing learning times. In the case of regular verbs only, the learning times trend more clearly upward, but they increase fairly slowly.

4.3 Discussion

In considering the results presented here, it is important to recognize that each of the three systems discussed in this paper have both strengths and weaknesses.

FOIDL provides consistent good accuracy, and can be very fast with small numbers of examples. It is important to note that FOIDL performs quite well on tasks where a top-down approach would be expected to do well (relatively few constants, smaller space of possible literals, fewer examples). Preliminary experiments with the finite mesh domain showed that FOIDL consistently outperformed BUFOIDL for that task, with at least comparable accuracy and better speed. However, FOIDL has significant problems dealing with larger search spaces.

CLOG was developed in response to this key problem with FOIDL, and, as a result, it is very fast. However, its two major drawbacks can be significant. First, the

decision list learned (and its quality) may depend heavily on the order in which examples are presented, since CLOG simply generalizes the first example in the training example set at each iteration. The learning algorithm does not seem to be as effective as the other two in producing decision lists with good predictive accuracy.

The second issue with CLOG is not a major drawback for the past tense task on which it was evaluated here, but could greatly limit the applicability of the approach. For the past tense task (and other similar tasks), it is fairly easy to determine what possible clauses can be constructed to generalize a given example. However, this is not the case for all tasks of potential interest. To construct the needed user-defined predicates would be an onerous task for some problems.

BUFOIDL is presented here as an alternative that does not share its predecessors' weaknesses; however, it is not a perfect answer. The system relies on the random selection of pairs of examples to generalize. If an insufficient number of examples is chosen at each iteration, BUFOIDL may fail to learn an accurate decision list. Of course, the system should not perform worse when selecting more pairs than required. The primary trade-off here is that the learning time is impacted by the number of pairs selected to learn from. Selecting many more pairs than are required will result in longer learning times than necessary.

A second weakness of BUFOIDL is its very long learning time for small training sets. Although BUFOIDL's learning time for larger training sets is clearly superior to FOIDL's, it takes far longer to learn from smaller training sets, since the training time is more closely tied to the number of pairs of examples chosen than to the number of examples in the training set. However, we perceive the necessity of spending minutes rather than seconds to learn from 25 examples well worth the advantage of learning from 1000 examples in minutes (rather than hours) as well.

Although BUFOIDL does not approach the lightning speed of CLOG, it clearly makes the learning of first-order decision lists with the level of accuracy that FOIDL provides a realistic possibility with larger example sets.

5 Related Work

The systems most closely related to BUFOIDL are CLOG and FOIDL. However, three other systems deserve mention.

Around the time of the development of FOIDL, Quinlan developed an alternate system for learning decision lists called FFOIL [12]. Quinlan's approach is based on FOIL, requiring extensional definitions of background predicates. While FFOIL does learn a decision list, it places the rules in the opposite order from BUFOIDL and FOIDL, so it does not take an approach of learning exceptions to previously learned rules. It does, however, learn a default rule that simply predicts the most common output and places that rule at the bottom of the decision list.

The TILDE system [1,2] is also very closely related to BUFOIDL. TILDE induces logical decision trees using a top-down approach that incorporates Blockeel and De Raedt's method of *learning from interpretations*. Blockeel and De Raedt show that their binary logical decision trees are equivalent in expressiveness to first order decision lists. However, their approach is not easily applicable to problems such as the past tense task discussed in this paper for two reasons. First, their approach is a top-down approach that requires the specification of constants to be used in the definitions

(a problem FOIDL also suffers from). More importantly, the method of learning from interpretations is specifically focused on classification tasks, and this particular task is not easily transformed into a classification paradigm.

Another somewhat related area of work is transformation-based learning [3]. Transformation-based learning systems learn a list of rules, and each rule strives to correct the errors made by the previous rules. Thus, we can see similarities of concept between the approaches of first-order decision list learning and transformation-based learning, as both do learn lists of rules, from general to specific, and both focus on learning rules to handle exceptions to previously learned rules. However, transformation-based learning systems apply all of the learned rules in order, while decision lists apply only the first applicable rule. Thus, one would expect the decision lists systems to be faster. It remains to be seen whether either approach is more accurate than the other.

6 Conclusions and Future Directions

In this paper, we have presented a new approach for learning first-order decision lists and have shown that it provides considerable speed-up over the most accurate existing system for learning this representation, while also providing comparable accuracy. However, we have only applied the system to data that the existing top-down approach could handle. The purpose in developing such a system is, of course, to be able to apply this learning approach to data sets too large for FOIDL. Therefore, our major direction for future work is to attempt to apply BUFOIDL to appropriate tasks. We will be looking primarily at language tasks, since those seem to fit the decision list paradigm. We also hope to do some comparisons between transformation-based learning approaches to language learning and decision list approaches.

References

1. Blockeel, H., De Raedt, L.: Top-Down Induction of First-Order Logical Decision Trees. Artificial Intelligence 101 (1998) 285-297
2. Blockeel, H., De Raedt, L., Jacobs, N., Demoen, B.: Scaling up Inductive Logic Programming by Learning from Interpretations. Data Mining and Knowledge Discovery. 3 (1999) 59-93
3. Brill, E.: Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging. Computational Linguistics. 21 (1995) 543-565
4. Califf, M.E., Mooney, R.: Advantages of Decision Lists and Implicit Negatives in Inductive Logic Programming. New Generation Computing. 16 (1998) 263-281
5. Califf, M.E., Mooney, R.:. Relational Learning of Pattern-Match Rules for Information Extraction. In Proceedings of the Sixteenth National Conference on Artificial Intelligence. AAAI Press Menlo Park, CA (1999) 328-334
6. Clark, P., Niblett, T.: The CN2 Induction Algorithm. Machine Learning, 3 (1989) 261-284
7. Manandhar, S., Džeroski, S., Erjavec, T.: Learning Multilingual Morphology with CLOG. In Proceedings of the 8th International Workshop on Inductive Logic Programming. Springer-Verlag Berlin Heidelberg New York (1998) 135-144
8. Mooney, R., Califf, M.E.: Induction of First-Order Decision Lists: Results on Learning the Past Tense of English Verbs. Journal of Artificial Intelligence Research. 3 (1995) 1-24

9. Muggleton, S., Feng, C.: Efficient Induction of Logic Programs. In Proceedings of the First Conference on Algorithmic Learning Theory. Tokyo, Japan (1990) 368-381
10. Muggleton, S.: Inverse Entailment and Progol. New Generation Computing. 13 (1995) 647-657
11. Quinlan, J.R.: Learning Logical Definitions from Relations. Machine Learning. 5 (1990) 245-286
12. Quinlan, J.R.: Learning First-Order Definitions of Functions. Journal of Artificial Intelligence Research. 5 (1996) 139-161
13. Rivest, R.L.: Learning Decision Lists. Machine Learning. 2 (1987) 229-246
14. Webb, G.I., Brkič, N.: Learning Decision Lists by Prepending Inferred Rules. In Proceedings of the Australian Workshop on Machine Learning and Hybrid Systems. Melbourne, Australia (1993) 6-10

Learning with Feature Description Logics

Chad M. Cumby and Dan Roth

Department of Computer Science
University of Illinois, Urbana, IL. 61801, USA
{cumby,danr}@uiuc.edu

Abstract. We present a paradigm for efficient learning and inference with relational data using propositional means. The paradigm utilizes description logics and concepts graphs in the service of learning relational models using efficient propositional learning algorithms. We introduce a *Feature Description Logic* (FDL) - a relational (frame based) language that supports efficient inference, along with a *generation function* that uses inference with descriptions in the FDL to produce features suitable for use by learning algorithms. These are used within a learning framework that is shown to learn efficiently and accurately relational representations in terms of the FDL descriptions.
The paradigm was designed to support learning in domains that are relational but where the amount of data and size of representation learned are very large; we exemplify it here, for clarity, on the classical ILP tasks of learning family relations and mutagenesis.
This paradigm provides a natural solution to the problem of learning and representing relational data; it extends and unifies several lines of works in KRR and Machine Learning in ways that provide hope for a coherent usage of learning and reasoning methods in large scale intelligent inference.

1 Introduction

In a variety of AI problems such as natural language understanding related tasks and visual inference there is a need to learn, represent and reason with respect to definitions over structured and relational data. Examples include the problem of identifying relations such as "A is the assassin of B", when attempting to answer a free-form question given potentially relevant text, identifying a seminar's speaker given the seminar's announcement, detecting people in an image or defining a policy that maps states and goals to actions in a planning situation.

In these cases it is natural to represent concepts relationally; propositional representations might be too large, could lose much of the inherent domain structure and consequently might not generalize well. In recent years, this realization has renewed the interest in studying relational representations both in the knowledge representation and reasoning (KRR) community and in the learning community. The challenge is to provide the expressivity necessary to deal with large scale and highly structured domains such as natural language (NL) and visual inference and at the same time meet the strong tractability requirements for these tasks.

This challenge has been a topic of active research in the knowledge representation and reasoning community for more than a decade. The main effort has been to identify classes

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 32–47, 2003.

of representations that are expressive enough to allow reasoning in complex situations yet are limited enough as to support reasoning efficiently [19,30]. It has become clear that propositional representations are not sufficient, and much effort has been devoted to studying languages that are subsets of first order logic, such as description logics and frame representation systems [1,6], as well as probabilistic augmentations of those [16].

The expressivity vs. tractability issue has been addressed also from the learning perspective, and a similar tradeoff has been observed and studied. While, in principle, Inductive Logic Programming (ILP) methods provide the natural approach to these tasks in that they allow induction over relational structures and unbounded data structures, theoretical and practical considerations render the use of unrestricted ILP methods impossible. These methods have also been augmented with the ability to handle uncertainty [14] although, as expected, this makes some of the computational issues more severe - studies in ILP suggest that unless the rule representation is severely restricted the learning problem is intractable [23,8,4,5]. The main way out of these computational difficulties has been via the use of propositionalization methods that attempt to learn classifiers for relational predicates via propositional algorithms, mapping complex structures to simple features [17].

This paper develops a paradigm for efficient learning and inference with structured data by building on progress made in both communities. This paradigm utilizes models developed in the KRR community, such as description logics and concepts graphs, in the service of learning relational models via efficient propositional learning algorithms.

We present a learning framework built around a *Feature Description Logic* (FDL) - a relational (frame based) language that supports efficient inference, along with an efficient *generation function* that uses inference with descriptions in the language in order to produce features suitable for use by learning algorithms.

In this paradigm, the description logic is an intermediate step, rather that the final representation, as is usual in KRR. The basic inference step, subsumption, is used as a means to transform a domain element, e.g., a natural language sentence, and represent it in terms of a richer vocabulary – descriptions in the FDL. This representation, in turn, may serve as an input to any propositional learning algorithm, including probabilistic algorithms, to yield structures in which sought after predicates are represented as functions (or conditional probabilities) over the relational descriptions.

In this respect our approach differs from standard ILP approaches and most propositionalization techniques. Features are generated up front before any learning stage in a data-driven way, allowing us to dictate the level of complexity of our intermediate representation before any search for a learned function occurs. Thus particularly expressive features that could not possibly be generated during the search itself are allowed to influence our final learned function in a significant way.

This paper provides a formal syntax and semantics for a specific feature description language (FDL). As in other description logics, we describe concepts in terms of *individuals* possessing attributes and roles in relation to other individuals. We then show the equivalence of descriptions in FDL to a class of concept graphs and use this to prove efficient subsumption between descriptions. We claim that the FDL language presented is only one member in a family of languages and that, in fact, several existing description languages, deterministic and probabilistic, can be used within our framework.

Our main construction is a Feature Generation Function that, given a FDL description and a domain element (e.g., an image, a sentence) represented as a relational structure, uses subsumption to re-represent it using a new vocabulary, in terms of the FDL descriptions. This representation, in turn, can be used also as a feature representation, usable by general purpose propositional learning algorithms.

Finally, after describing the general learning framework and the details of the FDL, we present an application of our technique to the "classical" ILP examples of learning family relationships and mutagenesis.

The problem of learning definitions for kinship relations has been an arch-typical example of relational learning since the late 80's/early 90's [12,24]. Simply put, given two members of a family, the task is to decide what their relation is. It has been argued that this task is a perfect example of a situation where relational learning is necessary, as both the concepts to be learned and all possible antecedents to a learned function are relations. In addition, ILP systems are purported to learn simple rules using a relatively small number of highly structured data instances. Thus it is an important task to test the viability of our relational learning paradigm.

Additionally we experiment with our technique on the ILP benchmark task of mutagenesis prediction, to which several propositionalization and general ILP methods have been applied [18,31]. We show that using our propositionalization technique we obtain respectable results in terms of the leading systems, with room for improvement.

While the learning approach is presented here as an approach to learn a definition for single predicates, we view this in a wider context. Learning definitions may be used to enrich vocabulary describing the input data; the FDL can then be used incrementally to produce useful features again and subsequently to build up new representations in terms of those in a manner similar to the one envisioned in [32]. Such a system might integrate easily into a programming platform, allowing researchers to construct large scale learning-based architectures to solve complex AI problems in areas such as natural language processing. It then becomes even more crucial that the basic components of this system are articulated in a language whose structure and meaning are well understood.

2 The Learning Framework

In this section we introduce a learning framework that can be used to generate classifiers for relational learning problems utilizing propositional learning algorithms. The key to our propositionalization technique is the combination of a feature extraction stage composing the data in a graph-based manner to produce features, and a classification stage utilizing a feature-efficient propositional learning algorithm to learn a function for each relation.

In our framework, as in ILP, observations in the domain are mapped into some collection of predicates that hold over elements in the domain. From this the task then becomes to produce a classifier to predict which predicates hold for some particular elements. For example, we may wish to predict that for some domain elements X and Y, the predicate $father(X, Y)$ holds. To accomplish this task using standard propositional learning algorithms, we must generate examples in the form of lists of active propositions (features) for each predicate to be learned. Propositions of this form may either be fully ground as in the predicate $father(john, jack)$, or individually quantified as in the

predicate $\exists X \ father(john, X) \wedge father(X, harry)$. Each list can also serve as a negative example for every other possible relation between elements that does not hold. These examples are used to train a classifier for each relation type, based on a linear function over the feature space. Our first major task then becomes to re-represent the data in a manner conducive to producing features for which we can learn a good discriminant function. Later we will show how by defining a focus of attention region "near" the predicate to be learned, we can differentiate the range of features produced based on the current target.

In order to facilitate this re-representation, we rely on a feature description language designed to model and work with the data in a graph-based manner. In theory and practice, this language is similar to the one developed in [7,26].

As we will see, the feature extraction method we present operates with the so-called closed-world assumption, generating only the features judged to be active in the example relative to the generating descriptions. All other features are judged to be inactive, or false. As it may be inefficient or impossible to list all features for a particular interpretation, this is a performance boon. Thus our learning algorithm should be able to accept examples represented as variable length vectors of only positive features.

In addition, our method provides the flexibility to generate a large number of features by designating a smaller set of "types" of features, so our learning algorithm should be able to learn well in the presence of a possibly large number of irrelevant features.

For the learning component, we use the SNoW learning system. SNoW[1] is a multi-class propositional classifier suited to a sparse representation of feature data of variable length and uses a network of linear functions to learn the target concept. It has been shown to be especially useful for large scale NLP and IE problems [15,26,11]. As SNoW employs a feature-efficient learning algorithm, Winnow [20], to learn a linear function over the feature space, it learns well in the presence of many irrelevant features. The linear threshold function also makes our hypotheses more expressive, as well as easier to learn.

2.1 Feature Description Logic

Here we describe our basic Feature Description Logic (FDL), by providing its formal syntax and semantics. We provide example descriptions from the family relations domain similar to those used later in the actual experiments.

As in most formal description logics, FDL descriptions are defined with respect to a set X of *individuals*. However, unlike most KL-ONE-like description logics, the basic alphabet for FDL descriptions includes *attribute*, *value*, and *role* symbols. We differentiate attribute from role descriptions and our basic primitive description is an attribute-value pair. We also allow a non-functional definition of attribute descriptions and role descriptions; thus an attribute describing an individual may take many values and a role describing an individual could take several different fillers.

Definition 1. *A FDL* description *over the* attribute *alphabet* $Attr = \{a_1, ..., a_n\}$*, the* value *alphabet* $Val = v_1, ..., v_n$*, and the* role *alphabet* $Role = \{r_1, ..., r_n\}$ *is defined inductively as follows:*

[1] Available at `http://L2R.cs.uiuc.edu/~cogcomp/`

1. *For an attribute symbol a_i, a_i is a description called a* sensor. *For some value symbol v_j, $a_i(v_j)$ is also a description, called a* ground sensor. *We also define a special* identity sensor *denoted* **&**, *which represents all individuals x.*
2. *If D is a description and r_i is a role symbol, then (r_i D) is a* role *description.*
3. *If $D_1, ..., D_n$ are descriptions, then (***AND** $D_1, ..., D_n$*) is a description. (The conjunction of several descriptions.)*

This definition allows for the recursive construction of complex FDL descriptions:

Example 1. We can imagine that one might want to describe the set of people who are named Charles, who are the grandfather of someone named Michael, and the father of someone aged 52 who is also the father of someone named Michael. They might use this description:

(AND name(Charles) (father (AND age(52) (father name(Michael))))
(grandfather name(Michael)))

We note that to avoid undecidability results known for subsumption in KL-ONE-like languages with unrestricted fillers for role descriptions [27], we omit constructions from our language designed to capture equality between individuals in the extensions of different subdescriptions. Feature value maps or SAME-AS type operators as found in CLASSIC-like languages allow this type of constraint, but, in languages in which roles may take an undetermined number of fillers, usually render subsumption between descriptions undecidable. Another type of equality constraint that might better suit our purposes in extracting useful features is a constraint over the values taken for particular attributes for different individuals. For example, in a natural language understanding application, we might wish to capture the fact that the plurality of the subject matches that of the main verb. Such constraints might also prove to make subsumption between descriptions difficult.

Luckily, if the range of possible values for such attributes is known, we can simulate this constraint simply by enumerating each possible case in which the constraint is satisfied with a description. The union of all those individuals described by each is then the set of individuals that satisfy the general constraint.

Example 2. In our natural language task, we want to describe the set of words that act as the subject of the sentence, and for which the plurality of the subject matches that of the main verb. We describe plurality with the attribute *number*, which can take the values *single*, and *plural*. We also assume that the *subject-of* role is defined for the subject and main-verb of the sentence. The following two descriptions then capture this set of words:

(AND number(single) (subject-of number(single)))
(AND number(plural) (subject-of number(plural)))

We now turn to the semantics of FDL descriptions. This discussion follows a model-theoretic framework similar to that laid out in [6,1]. This definition uses the notion of an interpretation [21], and that of an *interpretation function* which can be viewed as the function that encodes the information about domain. For a domain element z we denote by z^I its image under the interpretation function. Since it will be clear from the context, we use the same notation for an interpretation and an interpretation function.

Definition 2. *An interpretation I consists of a domain Δ, for which there exists an interpretation function I. The domain is divided into disjoint sets of individuals, X, and values, V. The interpretation function assigns an element $v^I \in V$ to each value v. It assigns a set of binary relations a^I over $X \times V$ to each symbol a in Attr, and a set of binary relations r^I over $X \times X$ to each symbol r in Role. The* extension *of a FDL description is defined as follows:*

1. *The extension of a sensor is defined as $a(v)^I = \{x \in X | (x, v^I) \in a^I\}$. The extension of an existential sensor a^I is $\{x \in X | \exists v^I \in V$ s.t. $(x, v^I) \in a^I\}$.*
2. *The extension of a role is defined as (r D)$^I = \{x \in X | \forall y (x, y) \in r^I \rightarrow y \in D^I\}$.*
3. *The extension of a conjunctive expression (AND D_1 D_2) is defined as $D_1^I \bigcap D_2^I$.*

We can now define the *subsumption* of a FDL description D_1 by another description D_2. We say that D_1 *subsumes* D_2 iff the extension of D_2 is a subset of the extension of D_1. i.e. $D_1^I \supseteq D_2^I$ for all interpretations I. To show that FDL allows efficient subsumption, we use the notion of a concept graph that we define next.

2.2 Concept Graphs

The notion of concept graphs stems from work in the semantic network and frame-based representation conceived of in the late 1970's and early 1980's. In many ways description logics were invented to provide a concrete semantics for the construction of such graph-based knowledge representations. In our framework they provide a tool for computing subsumption between descriptions and as a convenient representation for learning examples in our learning framework.

FDL concept graphs are a variation on the type invented for [1] to explain "basic CLASSIC". A FDL concept graph is a rooted labeled directed graph $G = G(N, E, v_0, l_N(*))$, where N is a set of nodes, $n_0 \in N$ is the root of the graph, $E \subseteq (N \times N \times Role)$ a set of labeled edges (with role symbols as labels) and $l_N(*)$ is a function that maps each node in N to a set of sensor descriptions associated with it.

The semantics of FDL concept graphs is defined similarly to that of basic CLASSIC, minus those associated with equality constraints. The extension of a node in the graph is intended to be the set of individuals described by its corresponding description.

Definition 3. *Given a FDL concept graph $G = (N, E, n_0, l_N(*))$, a node $n \in N$, and an interpretation I in some domain Δ composed of elements X and values V, we say that an individual $x \in X$ is in the extension of n iff:*

1. *For each sensor $a_i(v) \in l_N(n)$, $a_i^I(x, v^I)$ is true. For each sensor $a_i \in l_N(n)$, $\exists v^I \in V$ s.t. $a_i^I(x, v^I)$ is true.*
2. *For each edge $(n, m, r_i) \in E$, $\forall y \in X$ if $r_i^I(x, y)$ then y is in the extension of m.*

As in [6,1], an individual x is in the extension of G, iff it is in the extension of n_0. It will be clear later that in our paradigm we care about extension only as a clean way to define subsumption; the more basic notion here is the description itself.

Example 3. Fig. 1 shows the concept graph for the description in Ex. 1.

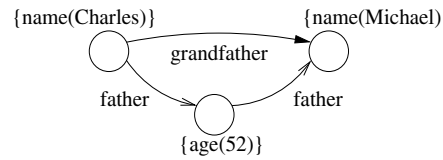


Fig. 1. The concept graph for the description in Ex 1.

Two constructs over domain Δ are semantically equivalent if they have the same extensions given an interpretation I. The significance of concept graphs for our purposes stems from the following theorem.

Theorem 1. *Any FDL description D is semantically equivalent to an acyclic FDL concept-graph of size polynomial in $\|D\|$ that can be constructed in polynomial time.*

Proof. We can recursively construct a concept graph for any FDL description in the following way, similar to the algorithm of [6,1]:

If the description is some sensor a_i or $a_i(v_j)$, construct a node labeled with the set $\{a_i\}$ or $\{a_i(v_j)\}$ respectively.

If the description is a role of the form $(r_i\ D)$, first recursively construct the concept graph for D. Then add an empty node n connected to the root of D's graph by an edge labeled r_i. Finally make n the root of the modified graph.

If the description is of the form (AND $D_1 \ldots D_n$), recursively construct the concept graphs for $D_1, \ldots D_n$. Then take the union of the node and edge sets for the above graphs, and merge the roots of all graphs. To merge two nodes $n_1\ n_2$, construct a new node n_{12} and add the union of the labels of n_1 and n_2 to it. Then redirect all the edges entering or leaving n_1 and n_2 to n_{12}. After all roots have been merged, this new node becomes the root of the combined graph.

The result of this polynomial time procedure is a semantically equivalent acyclic concept graph of size polynomial in the number of clauses in the original description, as each clause constructs at most a single node and edge.

Thm 1 allows now to show that FDL supports efficient subsumption queries between descriptions, and moreover that it supports checking subsumption of an arbitrary concept graph by a description.

Theorem 2. *For FDL descriptions D_1, D_2 the subsumption of D_2 by D_1 ($D_1 \supseteq D_2$) can be decided in polynomial time. Additionally for a description D_1 and an arbitrary FDL concept graph G_2, the subsumption of G_2 by D_1 can be decided in polynomial time.*

Proof. In order to compute subsumption of D_2 by D_1, first convert D_2 to a concept graph G_2. Then, from the root clause of D_1 recursively check each clause against G_2 in the following manner: Set the current clause c to the root clause in D_1 and the current node n to n_0 in G_2. If c is a sensor of the form $a_i(v_i)$, test if $a_i(v_i)$ is in $l_N(n)$. If c is a sensor of the form a_i, test if a_i is in $l_N(n)$, or if some $a_i(v)$ is in $l_N(n)$ for any value v.

If c is a role of the form $(r_i\ D)$, test if an edge labeled r_i from n to some n' exists in G_2. If so set n to n' and c to the clause D and recurse. If the check returns false for

D subsuming node n', but other edges labeled r_i exist from n, repeat the check for each node pointed to from n. Return true if any check returns true, else return false.

If c is a clause of the form (AND $D_1 \ldots D_n$), set c to each clause D_i and recursively test it on n. If all tests return true, then return true, else fail.

If the test for the root clause of D_1 returns true, then D_1 subsumes D_2. This test is obviously linear in the size of D_2 or in the size of a given G_2. If G_2 is given at the beginning as an arbitrary concept graph, the proof still holds. Note that while G_2 may then be cyclic, the D_1 description by definition is acyclic, so the procedure must end.

In our framework subsumption is used to transform a domain element, represented as a concept graph, into a feature set that can serve as an input to a propositional learning algorithm. The efficiency of this operation is crucial to support our efficient learning of relational representations. This result should be contrasted with general hardness results for subsumption [13,29].

Given these definitions for FDL descriptions and their corresponding concept graph representations, it now becomes possible to describe a feature extraction framework in which such representations play a major role. Efficient subsumption testing will allow the generation of expressive propositional features from arbitrarily complex data represented by concept graphs.

2.3 Feature Generating Functions

Up until this point, our treatment of our FDL has closely mirrored that of similar CLASSIC-like DL's [16,6,1]. However, our usage of FDL descriptions is vastly different from the usage of descriptions in these other DL's. The most closely related usage may be that of P-CLASSIC descriptions, in which a probabilistic distribution over descriptions is used to perform probabilistic subsumption queries. Instead, in our paradigm, descriptions are used to generate propositional formulae, in a data-driven way via subsumption queries. General propositional learning algorithms can then be used to learn definitions (classifiers) for predicates in terms of these or to learn distributions over them. We first describe the process of generating propositional formulae using FDL descriptions.

The essential construction of our method is that of a *Feature Generating Function* (FGF), closely related to the RGF of [7]. Our FGF construction, however, differs in an important respect. Here we discuss a general function, whose operation is constrained by the formal syntax of the generating descriptions themselves, having well defined structure and meaning. The FGF notion therefore extends and unifies the related constructions that have used a "relational calculus" to procedurally compose different types of RGFs in order to produce complex features. In fact, we claim that any operation of such a calculus may be pushed onto the syntax of an FDL, and therefore it is possible to define descriptions in our language that produce exactly the same features as produced there.

Definition 4. *Let I be some interpretation with domain $\Delta = (X, V)$, and let $\mathcal{I}$ be the space of all interpretations. For a description D we define a* feature F_D *to be a function $F_D : \mathcal{I} \rightarrow \{0, 1\}$. F_D acts an indicator function over $\mathcal{I}$, denoting the interpretations for which the extension of the description D is not empty.*

Example 4. A feature can be constructed from any FDL description. E.g., given the sensor description $D = name(Thelma)$, F_D might take an interpretation I in which for $x \in I$, $name(x, Thelma) = true$. Thus $F_D(I) = true$.

Given an interpretation I we say that a feature F is *active* in I if it evaluates to true. Such features could be a useful input to a propositional learning algorithm, as they describe hopefully expressive *relational* qualities of the given instance. Generating such features efficiently however is the topic of much debate, as such feature spaces could be prohibitively large or in some cases infinite, making manual generation impossible.

Our next step is to automate the construction of features of this sort. Luckily, the semantics of FDL descriptions and their equivalence to concept graphs gives rise to an efficient method of constructing *active* features, via the notion of the *feature generating function*.

Let some interpretation I be represented as a concept graph G, in which each element of I is in the extension of some node of G. To facilitate this construction, we define the inverse of the interpretation function associated with I. This inverse function is defined to output the value symbol v associated with each v^I, and to output the attribute or role symbol associated with each binary relation over elements and values. We process the interpretation by creating a node in G for each element seen in some relation, and processing each relation accordingly. If the relation corresponds to a role symbol, we create an edge between the nodes corresponding to the elements of the relation and label it with that role symbol. If the relation is of the form $a(x, v)$, with a corresponding to some attribute symbol a', we add a sensor description $a'(v')$ to the set $l(x')$ with the node x' corresponding to the element x in the relation. In this way, we create an FDL concept graph G with each element of I in the extension of a node.

Our FGF method takes G along with a set of input FDL descriptions $\mathcal{D}$, and outputs a set of active features over G. The basic method computes a feature description D_θ for G with respect to each description $D \in \mathcal{D}$, and constructs a feature for each D_θ. The intuition is that each input description defines a "type" of feature, subsuming many possible (partially) ground descriptions over several different interpretations. We say a description is *ground* if it is a description containing only sensors of the form $a_i(v_i)$.

Definition 5. *Let $\mathcal{F}$ denote an enumerable set of features over the space $\mathcal{I}$ of interpretations and let D be a description. A* feature generating function *$\mathcal{X}$ is a mapping $\mathcal{X} : \mathcal{I} \times D \rightarrow 2^{\mathcal{F}}$ that maps and interpretation I to a set of all features in $\mathcal{F}$ such that $F(I) = 1$ as constrained by the description D.*

Thus, the image of I under $\mathcal{X}$ is a re-representation of I in terms of the (set of D_θ's subsumed by the) description D.

Definition 6. *The* feature description *of a rooted concept graph G with respect to an input description D is defined to be the unique ground description D_θ subsuming G and subsumed by D, containing only ground forms of the sensors in D.*

In the case that D is itself ground, computing the feature description D_θ amounts to checking the subsumption of G by D. We note that our definition of the feature description for each input instance corresponds to only a single subsumer in the hierarchy of generality between the input description and that instance. In principle, this definition

could be expanded to the set of features corresponding to each point in this lattice of generality, in a manner similar to the version-space construction seen in [18].

Before discussing the mechanics of construction of each feature description, we give examples of features produced by our FGF.

Example 5. Given an input description D = **(AND name (father (AND age (father name)))**, the FGF $\mathcal{X}_{mss}(D, I)$ would produce the following active features for the family relations interpretation I defined in an earlier example. F_1*:(AND name(Charles) (father (AND age(52) (father name(Michael)))*. This feature represents the fact that in the current interpretation there is an individual whose name is Charles, who is the father of an individual with age 52 who, in turn is the father of some individual with the name Michael.

As usual, the importance of features stems from the fact that they might provide some abstraction. That is, they describe some significant property of the input which may occur also in other, different, inputs.

Theorem 3. *There exists a feature generating function $\mathcal{X}$ that, given any interpretation I represented as a concept graph and a description D, will generate all active features over I with respect to D in polynomial time.*

Proof. We provide an operational definition for $\mathcal{X}$ in terms of its operation on some interpretation I with respect to D. Given an I represented as a concept graph, we first designate one of the nodes n of I as the root, and construct the feature description of n as follows:

Test whether D subsumes n using the procedure defined above, copying each clause of D to a separate description D_θ in which each sensor is written as the version ground by the value from the corresponding node of I. If several values exist for the same sensor at the same node of I, construct a separate D_θ for each. If we find that D does not subsume n, we throw D_θ away. Then for each D_θ write an active feature F_{D_θ}. Repeat this procedure for each node n of I.

Since this procedure repeats the linear time subsumption procedure for FDL descriptions for each node in I, and in the worst case I and D may both have n nodes/clauses, the entire algorithm takes $O(n^2)$ time for each interpretation.

The importance of providing a formal framework for the feature generating process, via the FGF constructions stems from the fact that this notion generalizes essentially all ad-hoc ways in which people have been defining complex features in applications. For example, all "ngram"-like features that are used in NL applications with a variety of algorithms [22] can be easily defined via an FDL, and produced via the feature generating function construction. The current framework allows for the definition of significantly more expressive features, with a clear semantic and syntax, and with efficient inference procedure. As an example, there is a large body of literature in NLP on feature languages developed as an alternative, rich representation of natural language [3]. This very interesting but somewhat stagnant direction could be pushed forward if, for example, these languages are phrased as FDLs, and incorporated into the framework presented here via the notion of FGFs. This will allow learning and representing complex and probabilistic predicates in terms of descriptions in the languages.

3 Experiments and Results

We provide two experimental evaluations of our propositionalization technique: first in the task of learning kinship relations, and second in the domain of mutagenesis. Finally we present the results of both experiments with discussion.

3.1 Learning Kinship Relations

As mentioned earlier, the problem of learning a definition for kinship relations has been an arch-typical example of relational learning since the late 80's/early 90's [12,24]. Simply put, given two members of a family, the task is to decide what their relation is.

The data set used is the same set of 112 positive relations over two separate families originally used in [12], and later described in [24]. These relations are of the form *mother(Penelope,Arthur)* etc. and describe the kinship relationship of the first person to the second. In previous ILP experiments regarding this problem, a rule set is incrementally generated to cover more of the predicates seen in the data. In our experiments we take a markedly different approach, as we must formulate the data as examples for an on-line learning algorithm.

With a formal notion of concept graphs and descriptions in place, we can describe the setup of our learning experiments. We map all 112 relations of the aforementioned family relations data to a single interpretation I represented as a concept graph, where we define a node for each person in the data set and a directed edge for each relation.

In general, our learning system will process each interpretation, generating features as described earlier. Each feature is denoted by a description, serving as a unique lexical index. Each vector of features, called an *observation*, is passed to the learning algorithm, which treats one (or more) of them as a class label and the rest as features to be learned from. As features are generated in a data-driven way from a possibly infinite attribute space [2], observations will necessarily be variable-length vectors of features. Using the SNoW system allows us to learn effectively from this form of input.

To learn a definition for each target relation, we first generate an observation for each relation between two nodes in the data instance. This brings up the important notion of *focus of attention*. In many learning scenarios such as the family relations scenario, representing the data in a graph-based manner allows us to set a focus of attention based on the target predicate to be learned. This focus, for our purposes, takes the form of a node or a pair of nodes. The focus of attention allows us to restrict the range of nodes from which to produce features for the current observation based on certain graph properties in the interpretation - *e.g.* proximity to the focus. Thus, we can represent all of our family data as a single graph, and based on which nodes are targeted, can extract different meaningful features for each observation. The method we use to restrict features produced based on locality is described below.

In theory, our locality restriction is similar to the restriction made in [10] of features based on "context-dependent node attributes of depth n" and "context-dependent edge attributes of depth n". We restrict the range of features produced by restricting the set of nodes that can act as roots of features. This is accomplished by adding an operator to our standard FDL syntax as follows:

Given a description D and interpretation I, we say that the expression **(IN(n) D)** is a description denoting the set of individuals within n role edges from the focus of

attention of I. We then add an argument to our FGF function $\mathcal{X}$ to denote the focus of attention for I, and write it as $\mathcal{X}(I, D, (i, j))$, where D contains an **IN** clause and i and j are specifications of nodes in I. $\mathcal{X}$ is then evaluated in the following manner: If $i = j$ and n is positive, only generate features for nodes up to n edges away from i along some path leading from i. If n is negative then only generate features for nodes up to n edges away from i along some path leading *to* i. If $i \neq j$, then follow the same procedure, except for negative n test if it is along the path to i, and for positive n test if it is along the path from j. If $n = 0$, then only generate features for i along any path to j.

Now we describe the 3 sets of features generated for each observation. We extracted a feature for each single edge between the the focus nodes, corresponding to the class label, via the locality restriction measure defined above. We then extracted sets of features corresponding to each labeled path of size 2 and 3 between the focus nodes to serve as features. Our first set of input descriptions are of the form **(IN(0) (*rel* &))**, where rel is each of the 12 family relations in the data. Features produced from this description will serve as class labels in the observations. The second set is of the form **(IN(0) (*rel* (*rel* &)))** and the third of the form **(IN(0) (*rel* (*rel* (*rel* &))))**, where rel is again each of the 12 relations. These descriptions will produce the features from which the classifier will be learned.

We trained and tested the SNoW classifier with our relational features in a winner-take-all manner. Thus for each class label we obtain a linear function $\sum_i w_i \cdot F_i(I)$ of weighted features, with indices corresponding to lexical FDL descriptions. For example, F_i might be ***(IN(0) (brother (mother &)))***, or ***(IN(0) (husband (sister (father &))))***, which could be associated with the *uncle* class label.

We train and test only on pairs connected by valid relations, which correspond to the 112 positive examples. This method is similar to the approaches taken in [12,24,25] where only positive examples along with a handpicked set of negative examples were used in training and testing.

3.2 Mutagenesis

Mutagenicity prediction has become a standard benchmark problem for propositionalization methods in ILP [31]. In order to test the validity of our method, we trained classifiers for mutagenicity prediction on both the set of 188 "regression-friendly" compounds and the set of 42 "regression-unfriendly" compounds. For each compound, data is given in the form of tuples $atm(d2, d2\_1, c, 22, 0.0067)$ and $bond(d2, d2\_1, d\_2, 7)$ for the atom-bond information along with global information about the compound such as it's mutagenicity, lumo, and logP values.

We map each compound to a concept graph, constructing nodes for each atom tuple labeled with sensor descriptions of the form *atom-elt(v)* for the element type, *atom-type(v)* for the atom type *e.g. 22 = aromatic* , and *atom-chrg(v)* for the partial charge. We construct edges for each bond tuple labeled with the bond type, and also construct a single node for the compound overall, labeled with sensors of the form $active(v)$ for the compound's mutagenicity value, $lumo(v)$ for the compound's lumo value, and $logP(v)$ for the logP value. Lumo and logP values were discretized to a set of 10 ranges for each quantity.

To extract features, we then use the following input descriptions with the FGF: For the label features we use the simple description **active** which would then output features

with the lexical indices **active(true)** and **active(false)**. Features for the lumo and logP were extracted with similar descriptions. For the set of 188 compounds, all other features were extracted with input descriptions of the form:
(AND atom-elt atom-chrg (*bond-type* (AND atom-elt atom-chrg (*bond-type* ...))))
with the full description containing 9 conjunctions of atom and bond sub-descriptions.

For the set of 42 compounds in the "regression-unfriendly" dataset, a similar set of input descriptions was used, except in this case each contained a conjunction of 12 atom-bond sub-descriptions. After all features were extracted to form a set of propositional examples for each dataset, two classifiers were trained using the SNoW classifier, using the Winnow algorithm in each case.

3.3 Results and Discussion

For our kinship experiment, we compare our performance with the FOIL system of [24] and the relational pathfinding enhanced FORTE system presented in [25]. Training and testing sets were selected randomly using an incremental partitioning of the 112 positive examples. Accuracy was computed for each partitioning by averaging over 20 runs over a random selection of training and test examples. For the first experiment, after two cycles of training on 101 examples, our accuracy on the majority of runs was 100%, with an error on a few runs. These few errors were due to the test set not covering some relation because of the randomization, and on average over all the runs our accuracy converged to 99.36%.

By comparison, the FOIL system achieved an accuracy of 97.5% after performing 500 sweeps of 100 training examples and testing on the remainder averaging over 20 runs. The pathfinding version of the FORTE system also achieved 100% accuracy, but after training over a set of 300 randomly chosen positive and negative examples.

For the first mutagenesis dataset, the classifier was trained and tested using standard 10-fold-cross-validation, while for the second dataset a leave-one-out procedure was utilized. In the first case, for each fold, 12 cycles of training over the remaining data occurred before testing. In the second 20 cycles of training occurred for each training set. Using only the structural features defined above via the FDL, along with the discretized logP and lumo features, our system achieved an accuracy of 88.6% on the "regression-friendly" dataset of 188 compounds, and 86.3% on the "regression-unfriendly" dataset.

While the results shown on the mutagenesis problem for the primary dataset of 188 compounds do not measure up to the leading technique [28] (with an accuracy of 93%), they are on the level of the results achieved in [31] using the Progol system. Furthermore, our results were obtained without the use of any explicitly pre-encoded structural information, such as the indicator variables provided.

Our experiments showcase the ability of our system to match the output level of existing ILP systems, while exhibiting a drastic increase in efficiency both in learning and evaluation. Several factors contribute to this increase as well as to the high level of accuracy maintained.

First, as we have shown, efficient constructions for generating and evaluating formulae make feature construction easy. Instead of conducting a search for good bindings during the learning process, all active formulae are generated up-front as propositions, and feature efficient propositional algorithms take care of learning good definitions for

the functions. Second, the expressivity of our learned functions for these targets is actually *greater* than that of competing ILP methods. Due to the restrictions necessary for ILP methods to be tractable, rules learned in systems such as FOIL are defined over horn clauses, while our learned hypotheses are linear functions.

Supporters of the ILP paradigm also purport that a disjunctive set of rules is easier to interpret after learning than a network of features and weights. Yet in this case as all features are clauses similar to those induced by an ILP algorithm, it is a simple matter to examine the final learned weight vector to see what clauses contribute heavily to the learned function. For example, in the family relations experiment, for the target feature **(IN(0) (uncle &))** in our final weight vector the feature **(IN(0) (husband (aunt &)))** achieves a high weight, while the feature **(IN(0) (brother (mother &)))** has a slightly lower weight but still contributes to the final activation when the person corresponding to the intermediate node is not married.

4 Conclusion

We presented a paradigm for efficient learning and inference with relational data. This paradigm defined the notion of feature description logics - a relational language with clear syntax and semantics that can be used, via feature generation functions, to efficiently re-represent world observations in a way that is suitable for general purpose learning algorithms. We have shown that this allows one to efficiently learn complex relational representations, in a system in which the basic components are articulated in a language whose structure and meaning are well understood.

While we have concentrated on outlining the approach for a specific FDL, it is important to point out that a wide family of FDLs can be used within our framework. In fact, other CLASSIC-like description logics as well as their probabilistic variations, can be incorporated into the framework with the addition of a Feature Generating Function, and participate as building blocks in the process of learning relations and predicates. For example, features generated by FGFs need not be defined as Boolean. They can be associated with a real number, indicating the probability the feature holds in the interpretation, and can be used by the learning algorithm, allowing a immediate use of P-Classic like languages. This approach provides a different view on ways to extend such description languages - orthogonal to the one suggested by existing extensions, such as PRMs [9], that are more suitable to relational database-like (probabilistic) inferences and do not provide a natural solution to learning predicates and relational structure as in the examples shown here.

We have also shown the suitability of our framework for learning in purely relational domains by examining it in the context of the classical ILP problems of learning family relationships and mutagenesis. In these examples, we show the viability and flexibility of our propositionalization approach by exhibiting promising performance results both in terms of accuracy and speed. A variation of our framework has also been applied successfully to problems in the realm of Information Extraction.

This work extends and unifies several lines of works in KRR, learning and natural language processing, and provides ways to build on existing work in these areas and use it in ways that provide hope for a coherent usage of learning and inference methods for large scale intelligent inference.

References

1. P. F. Patel-Schneider A. Borgida. A semantics and complete algorithm for subsumption in the classic description logic. *J. of Artificial Intelligence Research*, 1:277–308, 1994.
2. A. Blum. Learning boolean functions in an infinite attribute space. *Machine Learning*, 9(4):373–386, 1992.
3. Bob Carpenter. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.
4. W. Cohen. PAC-learning recursive logic programs: Efficient algorithms. *Journal of Artificial Intelligence Research*, 2:501–539, 1995.
5. W. Cohen. PAC-learning recursive logic programs: Negative result. *Journal of Artificial Intelligence Research*, 2:541–573, 1995.
6. W. W. Cohen and H. Hirsh. Learnability of description logics with equality constraints. *Machine Learning*, 17(2/3):169–200, 1994.
7. C. Cumby and D. Roth. Relational representations that facilitate learning. In *Proc. of the International Conference on the Principles of Knowledge Representation and Reasoning*, pages 425–434, 2000.
8. S. Dzeroski, S. Muggleton, and S. Russell. PAC-learnability of determinate logic programs. In *Proceedings of the Conference on Computational Learning Theory*, pages 128–135, Pittsburgh, PA, 1992. ACM Press.
9. N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI*, pages 1300–1309, 1999.
10. P. Geibel and F. Wysotzki. Relational learning with decision trees. In *European Conference on Artificial Intelligence*, pages 428–432, 1996.
11. A. R. Golding and D. Roth. A Winnow based approach to context-sensitive spelling correction. *Machine Learning*, 34(1-3):107–130, 1999. Special Issue on Machine Learning and Natural Language.
12. G. E. Hinton. Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12, Amherst, Mass, August 1986.
13. D. Kapur and P. Narendran. NP-completeness of the set unification and matching problmes. In *Proc. of the 8th conference on Automated Ddeduction*, volume 230, pages 489–495. Springer Verlag, 1986.
14. K. Kersting and L. De Raedt. Bayesian logic programs. In J. Cussens and A. Frisch, editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 138–155, 2000.
15. R. Khardon, D. Roth, and L. G. Valiant. Relational learning for NLP using linear threshold elements. In *Proc. of the International Joint Conference of Artificial Intelligence*, 1999.
16. D. Koller, A. Levy, and A. Pfeffer. P-classic: A tractable probabilistic description logic. In *Proc. of the National Conference on Artificial Intelligence*, pages 360–397, 1997.
17. S. Kramer, N. Lavrac, and P. Flach. Propositionalization approaches to relational data mining. In S. Dzeroski and N. Lavrac, editors, *Relational Data Mining*. Springer Verlag, 2001.
18. S. Kramer and L. De Raedt. Feature construction with version spaces for biochemical applications. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML-2001)*, 2001.
19. H. Levesque and R. Brachman. A fundamental tradeoff in knowledge representation and reasoning. In R. Brachman and H. Levesque, editors, *Readings in Knowledge Representation*. Morgan Kaufman, 1985.
20. N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

21. J. W. Lloyd. *Foundations of Logic Progamming*. Springer-verlag, 1987.
22. L. Mangu and E. Brill. Automatic rule acquisition for spelling correction. In *Proc. of the International Conference on Machine Learning*, pages 734–741, 1997.
23. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 20:629–679, 1994.
24. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.
25. B. L. Richards and R. J. Mooney. Learning relations by pathfinding. In *National Conference on Artificial Intelligence*, pages 50–55, 1992.
26. D. Roth and W. Yih. Relational learning via propositional algorithms: An information extraction case study. In *Proc. of the International Joint Conference on Artificial Intelligence*, pages 1257–1263, 2001. Acceptance Rate: 197/796 (25%).
27. M. Schmidt-Schauss. Subsumption in KL-ONE is undecidable. In *Proc. of the International Conference on the Principles of Knowledge Representation and Reasoning*, pages 421–431, Boston (USA), 1989.
28. M. Sebag and C. Rouveirol. Tractable induction and classification in fol. In *Proceedings of IJCAI-97*, pages 888–892, 1997.
29. M. Sebag and C. Rouveirol. Any-time relational reasoning: Resource-bounded induction and deduction through stochastic matching. *Machine Learning*, 35:147–164, 1999.
30. B. Selman. *Tractable Default Reasoning*. PhD thesis, Department of Computer Science, University of Toronto, 1990.
31. A. Srinivasan, S. Mugleton, R. D. King, and M. Sternberg. Theories for mutagenicity: a study of first order and feature based induction. *Artificial Intelligence*, 85(1-2):277–299, 1996.
32. L. G. Valiant. Robust logic. In *Proceedings of the Annual ACM Symp. on the Theory of Computing*, 1999.

An Empirical Evaluation of Bagging in Inductive Logic Programming

Inês de Castro Dutra, David Page, Vítor Santos Costa, and Jude Shavlik

Department of Biostatistics and Medical Informatics and
Department of Computer Sciences
University of Wisconsin-Madison, USA

Abstract. Ensembles have proven useful for a variety of applications, with a variety of machine learning approaches. While Quinlan has applied boosting to FOIL, the widely-used approach of bagging has never been employed in ILP. Bagging has the advantage over boosting that the different members of the ensemble can be learned and used in parallel. This advantage is especially important for ILP where run-times often are high. We evaluate bagging on three different application domains using the complete-search ILP system, Aleph. We contrast bagging with an approach where we take advantage of the non-determinism in ILP search, by simply allowing Aleph to run multiple times, each time choosing "seed" examples at random.

1 Introduction

Inductive Logic Programming (ILP) systems have been quite successful in extracting comprehensible models of relational data. Indeed, for over a decade, ILP systems have been used to construct predictive models for data drawn from diverse domains. These include the sciences [16], engineering [10], language processing [33], environment monitoring [11], and software analysis [5]. In a nutshell, ILP systems repeatedly examine candidate clauses (the "search space") to find good rules. Ideally, the search will stop when the rules cover nearly all positive examples with only a few negative examples being covered.

Unfortunately, the search space can grow very quickly in ILP applications. Several techniques have therefore been proposed to improve search efficiency. Such techniques include improving computation times at individual nodes [4,26], better representations of the search [3], sampling the search space [27,28,32], and parallelism [8,13,19]. Parallelism can be obtained from very different alternative approaches, such as dividing the search tree, dividing the examples, or even through performing cross-validation in parallel [31].

An intriguing alternative approach that can lead to better accuracy whilst taking advantage of parallelism is the use of *ensembles*. Ensembles are classifiers that combine the predictions of multiple classifiers to produce a single prediction [9]. To some extent, an induced theory is an ensemble of clauses. We would like to go one step further and *combine different theories to form a single ensemble*. The main advantage is that the ensemble is often more accurate than its individual components. Moreover, we can use parallelism both in generating and in actually evaluating the ensemble.

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 48–65, 2003.

Several methods have been presented for ensemble generation. In this work, we concentrate on a popular method that is known to generally create a more accurate ensemble than individual components, *bagging* [6]. Bagging works by training each classifier on a random sample from the training set. In contrast to other well-known techniques for ensemble generation, such as boosting [12], bagging has the important advantage that it is effective on "unstable learning algorithms" [7], where small variations in parameters can cause huge variations in the learned theories. This is the case with ILP. A second advantage is that it can be implemented in parallel trivially.

We contrast bagging with a method we name *different seeds*, where we try to take advantage of the non-determinism in seed-based search by simply combining different theories obtained from experimenting with different seed examples, while always using the original training set. This method is also easily parallelisable.

Several researchers have been interested in the use of ensemble-based techniques for Inductive Logic Programming. To our knowledge, the original work in this area is Quinlan's work on the use of boosting in FOIL [25]. His results suggested that boosting could be beneficial for first-order induction. More recently, Hoche proposed confidence-rated boosting for ILP with good results [15]. Zemke recently proposed bagging as a method for combining ILP classifiers with other classifiers [34]. The contributions of our paper are to experimentally evaluate bagging on three particularly challenging ILP applications, and to compare bagging with the approach of different seeds.

The paper is organised as follows. First, we present in more detail ensemble techniques, focusing on bagging. Next, we discuss our experimental setup and the applications used in our study. We then discuss how ensembles perform on our benchmarks. Last, we offer our conclusions and suggest future work.

2 Ensembles

Ensembles aim at improving accuracy through combining the predictions of multiple classifiers in order to obtain a single classifier. Experience has shown that ensemble-based techniques such as bagging and boosting can be very effective for decision trees and neural networks [24,21]. On the other hand, there has been less empirical testing with classifiers as logic programs.

Figure 1 shows the structure of an ensemble of logic programs. This structure can also be used for classifiers other than logic programs. In the figure, each program $P_1, P_2, \ldots, P_N$ is trained using a set of training instances. At classification time each program receives the same input and executes on it independently. The outputs of each program are then combined and an output classification reached. Figure 1 illustrates that in order to obtain good classifiers one must address three different problems:

- how to generate the individual programs;
- how many individual programs to generate;
- how to combine their outputs.

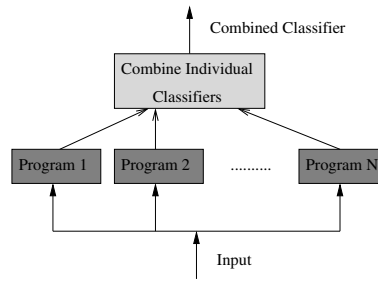


Fig. 1. An Ensemble of Classifiers.

Regarding the first problem, research has demonstrated that a good ensemble is one where the individual classifiers are accurate and make their errors in different parts of the instance space [17,22]. Obviously, the output of several classifiers is useful only if there is disagreement between them. Hansen and Salamon [14] proved that if the average error rate is below 50% and if the component classifiers are independent, the combined error rate can be reduced to 0 as the number of classifiers goes to infinity.

Methods for creating the individual classifiers therefore focus on producing classifiers with some degree of diversity. In the present work, we follow two approaches to producing such classifiers, described in the two following paragraphs.

One interesting aspect of Inductive Logic Programming is that the same learning algorithm may lead to quite different theories. More specifically, theories generated by seed-based ILP algorithms may heavily depend on the choice of the seed example. A natural approach to generate ensembles is to take advantage of this property of ILP systems, and combine the rather different theories that were generated just by choosing different sequences of seed examples. We call this approach *different seeds*.

We contrast the random choice of seeds with bagging. Bagging classifiers are obtained by training each classifier on a random sample of the training set. Each classifier's training set is generated by randomly, uniformly drawing K examples with *replacement*, where K is the size of the original training set. Thus, many of the original examples may be repeated in the classifier's training set.

Table 1 shows six training sets randomly generated from a set with examples numbered from 1 to 6. We can notice that each bagging training set tends to focus on different examples. The first training set will have two instances of the second and sixth examples, while having no instances of the second and fourth example. The second example will have instead two occurrences of the first and sixth example, while missing the second and third example. In general, accuracy for each individual bagging classifier is likely to be lower than for a classifier trained on the original data. However, when combined, bagging classifiers can produce accuracies higher than that of a single classifier, because the diversity among these classifiers generally compensates for the increase in error rate of any individual classifier.

Table 1. Example of Bagging Training Sets.

Training Sets	Examples Included					
1	6	2	6	3	2	5
2	1	4	6	5	1	6
3	1	3	3	5	2	3
4	6	4	1	4	3	2
5	6	4	2	3	2	3
6	5	5	2	1	5	4

Therefore, the promise of bagging, and of ensembles in general, is that when classifiers are combined the accuracy will be higher than the accuracy for the original classifier. In our case, one particularly interesting result for bagging is that it is effective on "unstable" learning algorithms, that is, on algorithms where a small change in the training set may lead to large changes in prediction. We focus on bagging in this work because it is considered to be less vulnerable to noise than boosting algorithms [12] and it can take advantage of parallel execution.

The second issue we had to address was the choice of how many individual classifiers to combine. Previous research has shown that most of the reduction in error for ensemble methods occurs with the first few additional classifiers [14]. Larger ensemble sizes have been proposed for decision trees, where gains have been seen up to 25 classifiers. In our experiments we decided to extend our analysis up to 100 classifiers.

The last problem concerns the combination algorithm. An effective combining scheme is often to simply average the predictions of the network [1,7,17,18]. An alternate approach relies on a pre-defined *voting threshold*. If the number of theories that cover an example is above or equal to the threshold, we say that the example is positive, otherwise the example is negative. Thresholds may range from 1 to the ensemble size. A voting threshold of 1 corresponds to a classifier that is the disjunction of all theories. A voting threshold equal to the ensemble size corresponds to a classifier that is the conjunction of all theories.

3 Methodology

We use the ILP system Aleph [29] in our study. Aleph assumes (a) background knowledge B in the form of a Prolog program; (b) some language specification $\mathcal{L}$ describing the hypotheses; (c) an optional set of constraints I on acceptable hypotheses; and (d) a finite set of examples E. E is the union of a nonempty set of "positive" examples E^+, such that none of the E^+ are derivable from B, and a set of "negative"examples E^-.

Aleph tries to find one hypothesis H in $\mathcal{L}$, such that: (1) H respects the constraints I; (2) The E^+ are derivable from B, H, and (3) The E^- are not derivable from B, H. By default, Aleph uses a simple greedy set cover procedure that constructs such a hypothesis one clause at a time. In the search for any

single clause, Aleph selects the first uncovered positive example as the seed example, saturates this example, and performs an admissible search over the space of clauses that subsume this saturation, subject to a user-specified clause length bound.

We have elected to perform a detailed study on three datasets, corresponding to three non-trivial ILP applications. For each application we ran Aleph with random re-ordering of the positive examples and hence of seeds. We call this experiment *different seeds*. Next, we created bagged training sets from the original set, and called Aleph once for each training set. We call this experiment *bagging*. The number of runs of *different seeds* is the same as the number of bags.

Aleph allows the user to set a number of parameters. We always set the following parameters as follows:

- search strategy: `search`. We set it to be breadth-first search, `bf`. This enumerates shorter clauses before longer ones. At a given clauselength, clauses are re-ordered based on their evaluation. This is the Aleph default strategy that favours shorter clauses to avoid the complexity of refining larger clauses.
- evaluation function: `evalfn`. We set this to be coverage. Clause utility is measured as $P-N$, with P and N being the number of positive and negative examples covered by the clause, respectively.
- chaining of variables: `i`. This Aleph parameter controls variable chaining during saturation: chaining depth of a variable that appears for the first time in a literal $\mathcal{L}_i$, is 1 plus the maximum chaining depth of all variables that appear in previous literals $\mathcal{L}_j, j < i$. We used a value of 5 instead of the default value of 2 in order to obtain more complex relations between literals in a clause.
- max number of nodes allowed: `maxnodes`. This corresponds to the number of clauses in the search space. We set this to be 100,000.
- maximum number of literals in a clause: `maxclauselength`. This was chosen to be the largest clause length that produced run times smaller than 1 hour on Intel 700 MHz machines, running Linux Red Hat 6.2. For our applications this value was either 4 or 5.

For each application, we ran experiments with different lower bounds on the minimum accuracy of an acceptable clause (`minacc`). We chose the values of 0.7, 0.9 and 1.0. In order to keep running times feasible, we first ran our datasets at least 5 times with the three different minaccs, and also with clause length varying from 4 to 10. We then chose the parameters that allowed Aleph to run at most for one hour (on Intel 700 MHz machines). It happened that for all minaccs, the clause length that produced runtimes less than or equal to one hour was the same, though it varied from one application to another. For each application we thus will vary our `minacc` settings among 0.7, 0.9 and 1.0, and we use the appropriate `maxclauselength`.

Next, we organise our discussion of methodology into (a) experimentation and (b) evaluation.

Experimentation. Our experimental methodology employs five-fold cross-validation. For each fold, we consider ensembles with size varying from 1 to 100.

The *minacc* parameter used to generate the component theories in an ensemble, and the voting threshold used for the ensembles, are tunable parameters. These parameters are tuned within each fold, using only the training data for that fold. Moreover, rather than holding out a single tuning set from the training data for a fold, we perform 4-fold tuning within the training set. The parameter combination that yields the highest accuracy during this "inner" 4-fold cross-validation on the training set is then used on the entire training set for that fold. The resulting theory is then tested on the test set for that fold. The process is repeated for each of the five folds, and the results are merged in the standard way for cross-validation.

Next, we present the details of the experimental setup, starting with tuning. As explained, our goal in the tuning phase is to estimate what is the best parameter setting (minacc, voting threshold) for the ensembles, in order to use them later during the training/test phase. Tuning proceeds in two steps. First, we repeatedly call Aleph to generate all the theories we need to construct the different ensembles. Because the ILP runs are time-consuming, we do not repeat the ILP runs themselves to learn entirely new theories for each different ensemble size. Rather, for each tuning fold and minacc parameter, we initially learn 100 theories using *different seeds* methodology and 100 theories using *bagging*. Then, for either ensemble approach, and for each ensemble size s between 1 and 99 we randomly select s different theories from the 100. We next use these theories to generate ensembles, and evaluate the results. Because our results may be distorted by a particularly poor or good choice of these theories, we repeat this selection process 30 times and average the results.

Tuning thus requires 12,000 theories per application: one theory for *bagging* plus one theory for *different seeds*, times 5 test set folds, times 4 tuning folds, times 3 minacc values, times the 100 different theories we create for producing ensembles.

Once the 12,000 theories are generated, two tables are produced. The first one, *minaccs_for_rocs*, is used to calculate the ROC curves and contains the best minacc for each ensemble size and for each voting threshold. The second one, *best_thresholds* is used to obtain the accuracies and contains the best combination of minacc and voting threshold for each ensemble size. This second table is a subset of *minaccs_for_rocs*.

We next move to the 5-fold cross-validation by doing a training/test per fold, per each minacc. First, we produce 600 theories per fold: one for bagging plus one for *different seeds*, times 3 minaccs, times the 100 different theories used to create ensembles. We are now ready to evaluate the ensembles.

Evaluation. For the evaluation phase (b), we used two techniques to evaluate the quality of the ensembles. First, we studied how average accuracy varies with ensemble size. We present accuracy as the average between accuracy on the positive examples and accuracy on the negative examples.

Second, we studied Receiver Operating Characteristic (ROC) curves for the ensembles. Provost and Fawcett [23] have shown how ROC curve analysis [20,35] can be used to assess classifier quality. When we consider the results of a partic-

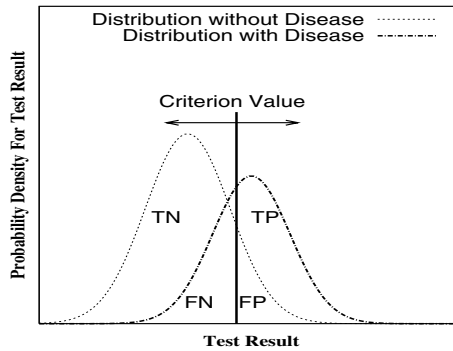


Fig. 2. Example of Probability Density Functions for Two Populations: with Disease, without Disease.

ular test in two populations, say positive and negative examples, we will rarely observe a perfect separation between the two groups. Indeed, the distribution of the test results can overlap, as shown in Figure 2. For every possible cut-off point or criterion value we select to discriminate between two populations, there will be some cases with the classifier correctly reporting positive examples to be positive (TP = True Positive fraction), but some cases incorrectly reported negative (FN = False Negative fraction). On the other hand, some negative examples will be correctly classified as negative (TN = True Negative fraction), but some negative examples will be classified as positive (FP = False Positive fraction).

In an ROC curve the true positive rate (sensitivity, or $\frac{TP}{TP+FN}$) is plotted against the false positive rate (100-specificity, or $\frac{FP}{FP+TN}$) for different cut-off points. Each point on the ROC plot represents a sensitivity/specificity pair corresponding to a particular decision threshold. A test with perfect discrimination (no overlap in the two distributions) has a ROC plot that passes through the upper left corner (100% sensitivity, 100% specificity). Therefore the closer the ROC plot is to the upper left corner, the higher the overall accuracy of the test [35].

When we have many ROC curves to be analysed, we can look instead to the area under those curves. The value for the area under the ROC curve can be interpreted as follows: an area of 0.84, for example, means that a randomly selected individual from the positive group has a test value larger than that for a randomly chosen individual from the negative group 84% of the time. When the variable under study cannot distinguish between the two groups, i.e. where there is no difference between the two distributions, the area will be equal to 0.5 (as is the case when the ROC curve coincides with the diagonal). When there is a perfect separation of the values of the two groups, i.e. there is no overlapping of the distributions, the area under the ROC curve equals 1 (the ROC curve will reach the upper left corner of the plot).

We wish to test the effectiveness of different sizes of ensembles, from 1 to 99. Again, we do not repeat the ILP runs themselves to learn entirely new theories for

each different ensemble size. Rather, we use the theories from the previous step. Because our results may be distorted by a particularly poor or good choice of these theories, we repeat this selection process 30 times and average the results.

Accuracies across the folds are obtained by averaging the sum of all positives and negatives for every fold. Areas across the folds are obtained by simply averaging areas computed for each ensemble size. ROC curves across the folds are obtained by averaging the rates of positives and negatives of each fold.

All experiments were performed using Condor, a tool for managing heterogeneous resources developed by the Condor Team at the UW-Madison [2]. Without the utilisation of such a tool, our experiments would have taken years to be concluded. Our jobs occupied about 53,380 hours of CPU, with an average peak of 400 jobs running simultaneously on Intel/Linux and Sun4u/Solaris machines.

3.1 Benchmark Datasets

Our benchmark set is composed of three datasets that correspond to three non-trivial ILP applications. We next describe the characteristics of each dataset with its associated ILP application, and present a dataset summary table.

Carcinogenesis. Our first application concerns the prediction of carcinogenicity test outcomes on rodents [30]. This application has a number of attractive features: it is an important practical problem; the background knowledge consists of large numbers of non-determinate predicate definitions; experience suggests that a fairly large search space needs to be examined to obtain a good clause.

Smuggling. Our second dataset concerns data on smuggling of some materials. The key element of our data is a set of smuggling *events*. Different events may be related in a variety of ways. They may share a common location, they may involve the same materials, or the same person may participate. Detailed data on people, locations, organisations, and occupations is available. The actual database has over 40 relational tables. The number of tuples in a relational table vary from 800 to as little as 2 or 3 elements.

The ILP system had to learn which events were *related*. We were provided with a set of related examples that we can use as positive examples. We can assume all other events are unrelated and therefore compose a set of negative examples. We assume *related* is comutative. Therefore we changed Aleph to assume `related(B,A)` if `related(A,B)` was proven, and vice-versa.

The smuggling problem is thus quite challenging in that it is a heavily relational learning problem over a large number of relations, whereas most traditional ILP applications usually require a small number of relations.

Protein. Our last dataset consists of a database of genes and features of the genes or of the proteins for which they code, together with information about which proteins interact with one another and correlations among gene expression

Table 2. Datasets Characteristics.

	Dataset Sizes	Max Clause Length
Carcinogenesis	182+/148-	4
Smuggling	143+/517-	5
Protein	172+/690-	5

patterns. This dataset is taken from the function prediction task of KDD Cup 2001. While the KDD Cup task involved 14 different protein functions, our learning task focuses on the challenging function of "metabolism": predicting which genes code for proteins involved in metabolism. This is not a trivial task for our ILP system.

Table 2 summarises the main characteristics of each application. The second column corresponds to the size of the full datasets, where `P+/N-` represents number of positive examples and number of negative examples. Bags are created by randomly picking elements, with replacement, from the full dataset. Therefore the number of positives or negatives of each bag are not the same as of the full dataset used for *different seeds*, although the total size is the same. The second column indicates the clause length used for each dataset. The test sets for each 5-fold cross-validation experiment is obtained by a block distribution of the full dataset. For example, application Carcinogenesis will have 5 positive test sets of sizes: 36, 36, 36, 36 and 38. These test sets are not used during the tuning phase.

4 Results

This section presents our results and analyses the performance of each application. For each application we show the average accuracy for positives and negatives, and the area under ROC curves built from 1 to 25 ensemble sizes. The results from 26 to 99 are essentially horizontal lines and are not shown. We report results for *different seeds* and *bagging*. We also show the ROC curve for an ensemble size of 25.

The accuracies presented in the graphs are averaged across all folds, and for each ensemble size, a different voting threshold and minacc are used. This combination of voting threshold and minacc is the one that produced the best accuracy during the tuning phase. For clarity's sake, these parameter values are not shown in the curves.

The areas under the ROC curves were computed by (1) computing an ROC curve, per fold, for each ensemble size using the theories learned for the best pair <*minacc, voting threshold*>, (2) computing the area under each ROC curve, and (3) averaging the areas for each ensemble size across the folds.

Figure 3 shows the average accuracies for the three applications, for *different seeds* and *bagging*, when varying the ensemble sizes. Figure 4 shows the areas under the ROC curves, averaged across five folds, for the three applications, when varying the ensemble sizes. Figure 5 shows ROC curves at ensemble size 25 for every application.

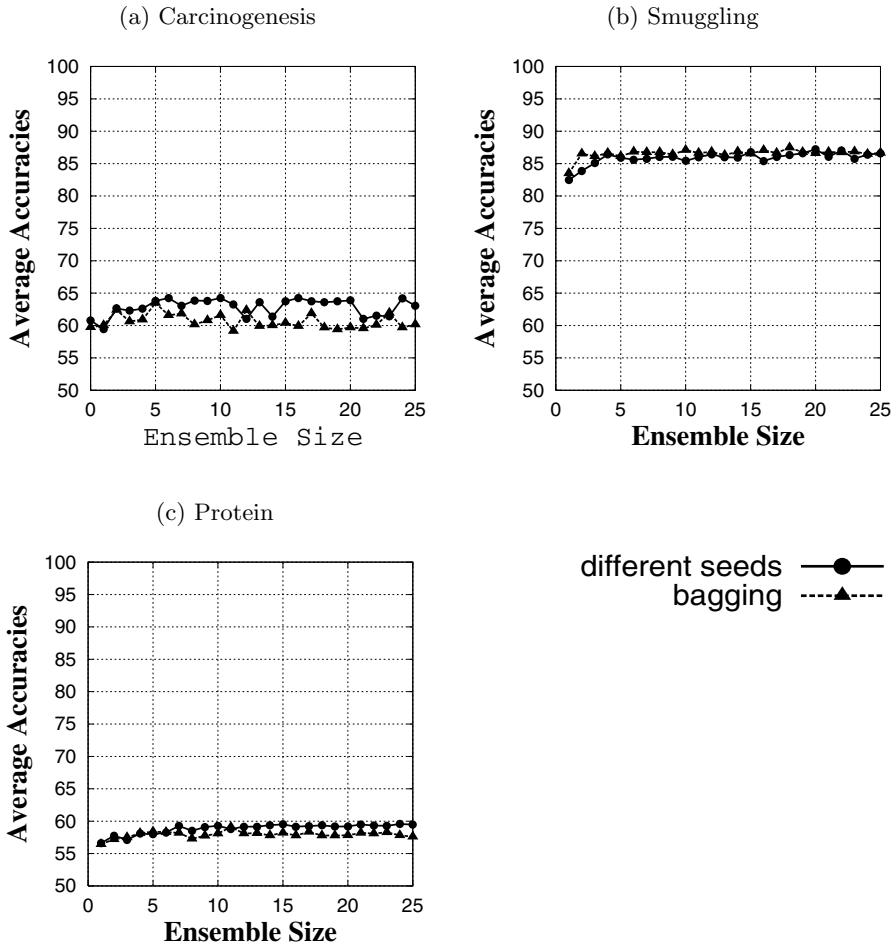


Fig. 3. Average Accuracies for the Three Applications.

The results show that ensembles do provide an improvement both in accuracies and in ROC areas. Most of the improvement is obtained for smaller ensemble sizes, up to 5 or 10 elements. Performance does not seem to benefit much from using larger sizes.

The best results were obtained in the smuggling application, with *bagging* and *different seeds* obtaining similar performance. The most irregular application is carcinogenesis. We discuss the individual applications in more detail in the next sections.

4.1 Carcinogenesis

Single-theory accuracy results for carcinogenesis are not very good. On average, accuracy for a single theory is around 60% for *different seeds* and 59% for

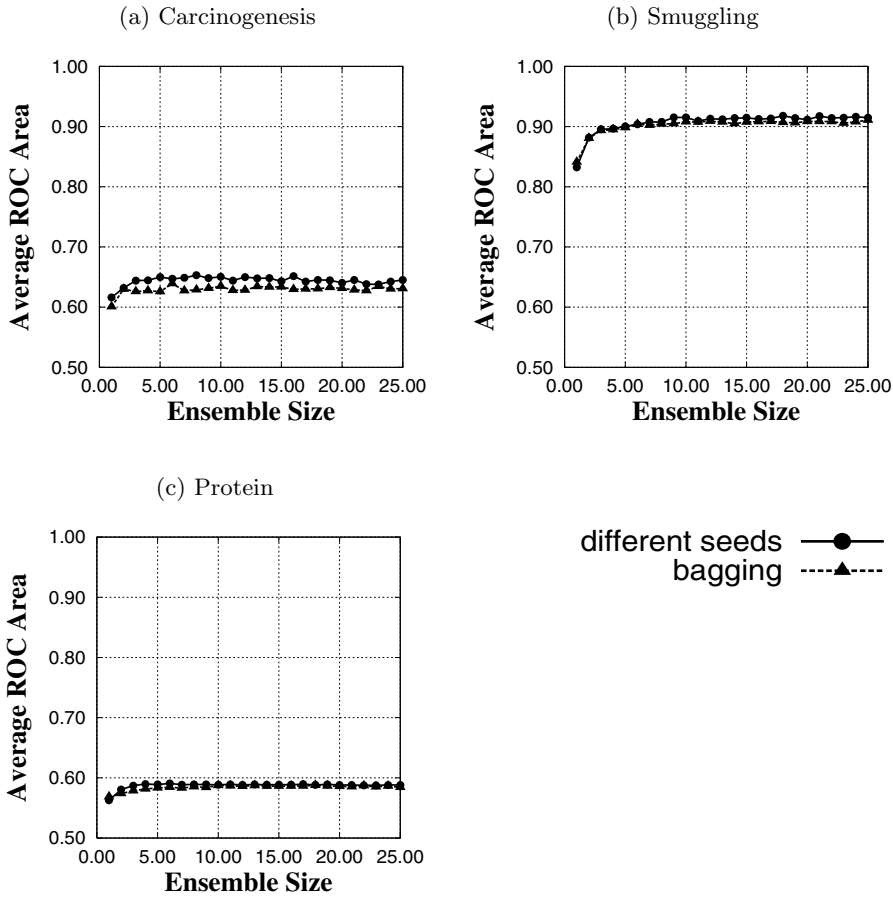


Fig. 4. Areas under the ROC curves for the Three Applications.

bagging. Our results show that ensembles can improve accuracy to around 64%. Unfortunately, our results also show huge variations, as we discuss next.

Figure 3(a) shows the average among the accuracy curves for the five folds. At first, the curves show that both *bagging* and *different seeds* obtain significant improvements. As ensemble size grows, *different seeds* tends to obtain better results, whereas *bagging* on average tends to achieve performance closer to the single-theory case. Both curves show a number of peaks.

The maximum average accuracy for *different seeds* is 64.5%, which represents a significant improvement over the single-theory accuracy. Studying *different seeds* in more detail, we found the system tends to be pretty reasonable at classifying positive examples. It achieves a maximum of 78.7% acceptance rate for *different seeds*, at ensemble size 31. It does not perform so well at rejecting negative examples: the best result is a minimum probability of 48.3% of rejecting a negative example, at ensemble size 12.

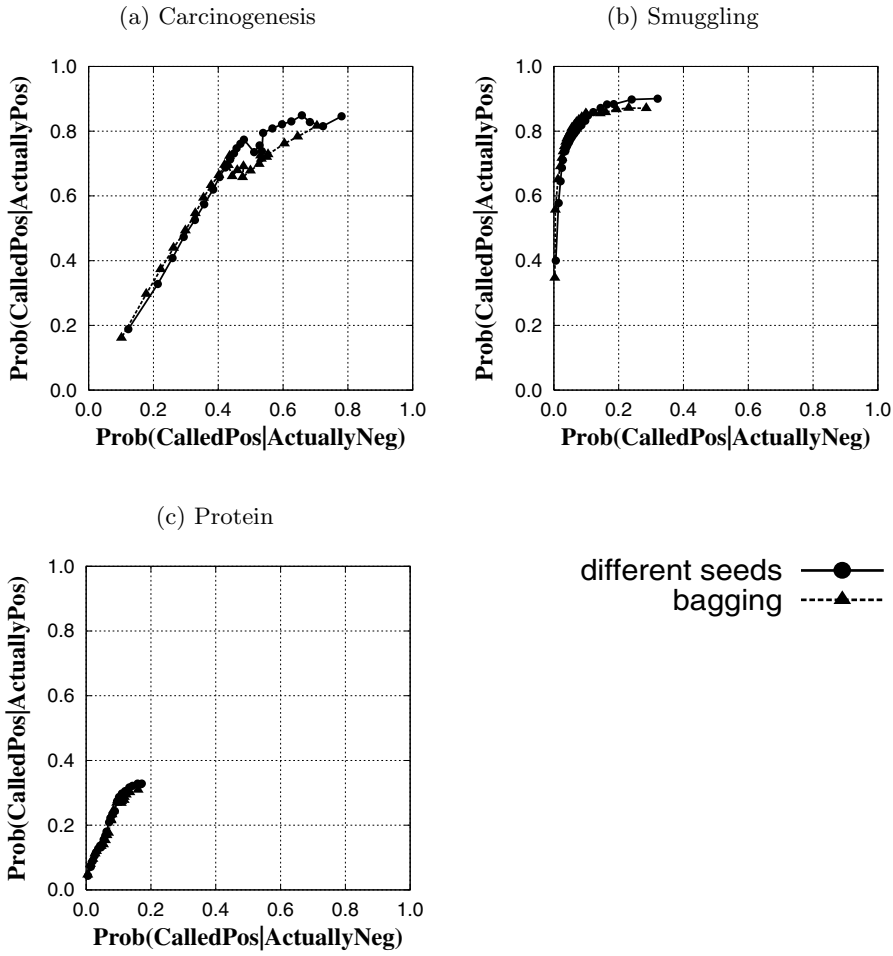


Fig. 5. ROC curves at N=25 for the Three Applications.

Different seeds is particularly interesting in that it shows several spikes, which are most obvious in the accuracy curve. This effect is caused by our tuning algorithm that has some difficulties with a very unstable application, such as carcinogenesis. More precisely, study of the data shows that often the tuning algorithm will choose the same minacc in a row for a series of consecutive ensemble sizes in a fold, and then all of a sudden select a different minac, and immediately return to choosing the initial parameter. The points where there is an abrupt change of parameters are usually the points where we have spikes.

The accuracy curve for *bagging* has several spikes on smaller ensembles, and then stabilises rather quickly. Again, most of the variation is caused by the choice of parameters. Interestingly enough, whereas most spikes in *different seeds* are negative, in this case most spikes are improvements. This suggests that accuracy

results may be suffering from a wrong choice, either of thresholds or of Aleph's minimal training accuracy parameter.

Next, we look at the areas under the ROC curves as ensemble size varies from size 1 to 25. The area under a ROC curve gives a good estimate of classifier quality.

Figure 4(a) compares the ROC areas for *different seeds* and for *bagging*. We can notice that both techniques obtain a significant improvement with the smaller ensembles. *Bagging* stabilises very quickly, though, whereas *different seeds* obtains improvements up to ensemble size 5. As a result, *different seeds* has a somewhat better result than *bagging*.

ROC curves provide a more detailed picture of the behaviour of this application concerning rates of true positives against false positives. We chose to plot an ROC curve for ensemble size of 25, the largest ensemble size for which we present other results. Figure 5(a) shows an ROC curve for ensemble size 25, for the application Carcinogenesis, for *different seeds* and *bagging*, when varying the voting threshold from 1 to 25. The curves show very clearly the spikes caused by the different minacc chosen for each point.

The curves show large variations for small ensemble sizes. This corresponds to choosing different minaccs. Both *different seeds* and *bagging* can achieve a very good improvement on positive examples. However, at a cost of increasing the rate of false positives. The best result for positives, for *different seeds* is achieved at voting threshold 1, with acceptance rate of 88%. *Bagging* also achieves its best performance on positives at threshold 1, but it performs a little worse than *different seeds* achieving maximum of 85% of acceptance rate. As the voting threshold increases, the chance of better classifying a positive or negative example decreases. Note that the voting threshold increases as we decrease along the X axis, but not in a consecutive order of points in the ROC curve. For example, at ensemble size 9, the false positive rate is 0.66, and the true positive rate is 0.81, while at ensemble size 10, the false positive rate is 0.70 and the true positive rate is 0.80.

Our main conclusion is that *different seeds* performs better for small thresholds, that is, it can recognise most positive examples. *Bagging* tends to perform better for large thresholds, that is, it is better at recognising negative examples.

4.2 Smuggling

The smuggling problem is quite challenging in that it is a heavily relational learning problem over a large number of relations, whereas most traditional ILP applications usually require a small number of relations. It was therefore quite interesting to find that Aleph can achieve quite good accuracy for this problem. We found average accuracy to be about 82% for *different seeds* and 83% for *bagging*, for a single theory. Single accuracy for negative examples for a single theory is around 89% for *different seeds* and 91% for *bagging*, while accuracy for positive examples are around 76%, for both *different seeds* and *bagging*.

Of course, it would be quite nice to achieve an even better accuracy. Figure 3(b) shows the variation of accuracy averaged across folds, as we range the

size of the ensembles between 1 and 25. Accuracy for both curves increases quickly for smaller ensembles and then is largely stable as we increase ensemble size. *Different seeds* and *bagging* achieve a maximum average accuracy of around 87%, with *different seeds* behaving slightly better than *bagging* for larger ensemble sizes. Our results thus correspond to a significant improvement over the single-theory accuracy. Accuracies stabilise at around 92%, for both *different seeds* and *bagging*, at classifying negative examples, and at around 82% for both *different seeds* and *bagging*, at classifying positive examples.

This application illustrates the benefit of using *bagging* at smaller ensemble sizes, and stresses the advantage of using ensemble methods to improve the accuracy of a single theory.

Figure 4(b) shows the areas under the ROC curves from ensemble size 1 to 25 averaged across the five test set folds. The results on ROC areas are very impressive. Performance is excellent for both *different seeds* and *bagging*, with a small advantage for *different seeds*. Both curves show a substantial improvement up to size 10, and then stabilise. Also notice that *bagging* and *different seeds* achieve very similar results.

Figure 5(b) shows average of ROC points across five folds, for ensemble size 25 varying the threshold from 1 to 25. Both classifiers perform in much the same way. The combined classifier is very good at classifying negative examples: even in the worst case, it only misclassifies up to 30% of all negative examples. Results are also quite good on the positive examples, although some examples are never covered. *Different seeds* does have some advantage in this case. Last, notice that there are much less spikes than for Carcinogenesis: the tuning algorithm seems to perform quite well here.

4.3 Protein

The Protein application has average single-theory accuracy of around 56% for both *different seeds* and *bagging*. Figure 3(c) shows the average accuracies between positives and negatives for *different seeds* and *bagging*, as we increase the ensemble size from 1 to 25. *Bagging* and *different seeds* have very similar performance for this application. The improvement over the single-theory is between 2 and 3 percentage points, and does show that the ensemble methods can improve performance even in this case.

Different seeds achieves maximum average accuracy of 60% at ensemble size 47, while *bagging* achieves maximum accuracy of 59% at ensemble size 28.

In order to understand better what happens with this application we draw the ROC curve for ensemble size 25. Figure 5(c) shows the ROC curves for *different seeds* and *bagging* averaged across five folds. The performance of this application on positives improves as we increase the ensemble size for both *different seeds* and *bagging*. For low thresholds, *different seeds* does better for positives. The learned theories seem to have difficulty in generalising: we cannot cover most examples. This results in bad accuracy for positives, and in good accuracy for negatives. The results also show that most of the improvement happens for smaller ensembles.

Our analysis provides more insight when we look at the areas under the ROC curves, varying our threshold from 1 to 25. Figure 4(c) shows the areas under the ROC curves. The results essentially confirm what we obtained with accuracy. *Different seeds* and *bagging* have the same performance up to ensemble size 24. After that *different seeds* surpasses the performance of *bagging*.

As with the other applications, both ensemble methods improve performance over the single-theory.

5 Conclusions and Future Work

This work presents an empirical study of *bagging*, a well-known ensemble building mechanism, in the Inductive Logic Programming setting. In our approach, we use bagging to combine theories built from random variations of the original training set. We contrast bagging to *different seeds*, an approach where we always use the same training set, and randomly select different seeds to build the different theories in the ensemble. We evaluated *bagging* and *different seeds* with three non-trivial applications of ILP.

Our results show that ensembles built through *bagging* can indeed achieve a sizable improvement in performance, both measured through accuracy and through ROC curves. Most of the gain is achieved with ensembles of size up to 20. Exploiting different seeds can also achieve very good gains. In fact, simply using *different seeds* worked as well, or arguably better, than *bagging* in our experiments. We believe this is because *different seeds* learns theories using the whole set of examples. Our results confirm the advantages of using ensemble methods in ILP.

We believe that *bagging* and *different seeds* can have a substantial impact on ILP. We can often achieve an interesting improvement in performance, with little implementation work. Moreover, we found that accuracy may improve, even in the cases where ILP is obtaining very good results, as is the case of the dataset Smuggling. On the other hand, resulting theories are more complex and thus harder to understand.

We have thus far used *bagging* and *different seeds* with ensembles of theories. An interesting alternative we are researching is to use ensembles of clauses. As discussed before, *boosting* can also be employed to improve accuracy of classifiers by penalizing examples that are misclassified. One disadvantage of *boosting* is that it can not be as easily parallelisable as *bagging* or *different seeds*. We have been investigating a method to perform *boosting* in parallel. Last, we are investigating other tuning algorithms to improve the interaction between different settings (e.g., clause length, minimum clause accuracy) for the ILP search and the *bagging*/*different seeds* process.

Acknowledgments

This work was supported by DARPA EELD grant number F30602-01-2-0571, the NSF grant 9987841 and by NLM grant NLM 1 R01 LM07050-01. Vítor Santos

Costa and Inês de Castro Dutra were partially supported by CNPq. We would like to thank the Biomedical Group support staff for their invaluable help with Condor. We also would like to thank Ashwin Srinivasan for his help with the Aleph system and the Carcinogenesis benchmark. Vítor Santos Costa and Inês Dutra are on leave from COPPE/Sistemas, Federal University of Rio de Janeiro.

References

1. E. Alpaydin. Multiple networks for function learning. In *IEEE International Conference on Neural Networks*, pages 9–14, 1993.
2. J. Basney and M. Livny. Managing network resources in Condor. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9), Pittsburgh, Pennsylvania*, pages 298–299, Aug 2000.
3. H. Blockeel, L. Dehaspe, B. Demoen, G. Janssens, J. Ramon, and H. Vandecasteele. Executing query packs in ILP. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 60–77. Springer-Verlag, 2000.
4. H. Blockeel, B. Demoen, G. Janssens, H. Vandecasteele, and W. Van Laer. Two advanced transformations for improving the efficiency of an ILP system. In J. Cussens and A. Frisch, editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 43–59, 2000.
5. I. Bratko and M. Grobelnik. Inductive learning applied to program construction and verification. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 279–292. J. Stefan Institute, 1993.
6. L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.
7. L. Breiman. Stacked Regressions. *Machine Learning*, 24(1):49–64, 1996.
8. L. Dehaspe and L. De Raedt. Parallel inductive logic programming. In *Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, 1995.
9. T. Dietterich. Ensemble methods in machine learning. In J. Kittler and F. Roli, editors, *First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2000.
10. B. Dolšak and S. Muggleton. The application of ILP to finite element mesh design. In S. Muggleton, editor, *Proceedings of the 1st International Workshop on Inductive Logic Programming*, pages 225–242, 1991.
11. S. Džeroski, L. Dehaspe, B. Ruck, and W. Walley. Classification of river water quality data using machine learning. In *Proceedings of the 5th International Conference on the Development and Application of Computer Techniques to Environmental Studies*, 1995.
12. Y. Freund and R. Shapire. Experiments with a new boosting algorithm. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 148–156. Morgan Kaufman, 1996.
13. J. Graham, D. Page, and A. Wild. Parallel inductive logic programming. In *Proceedings of the Systems, Man, and Cybernetics Conference*, 2000.
14. L. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, October 1990.

15. S. Hoche and S. Wrobel. Relational learning using constrained confidence-rated boosting. In Céline Rouveirol and Michèle Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *Lecture Notes in Artificial Intelligence*, pages 51–64. Springer-Verlag, September 2001.
16. R. King, S. Muggleton, and M. Sternberg. Predicting protein secondary structure using inductive logic programming. *Protein Engineering*, 5:647–657, 1992.
17. A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. The MIT Press, 1995.
18. N. Lincoln and J. Skrzypek. Synergy of clustering multiple backpropagation networks. In *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 1989.
19. T. Matsui, N. Inuzuka, H. Seki, and H. Ito. Parallel induction algorithms for large samples. In S. Arikawa and H. Motoda, editors, *Proceedings of the First International Conference on Discovery Science*, volume 1532 of *Lecture Notes in Artificial Intelligence*, pages 397–398. Springer-Verlag, December 1998.
20. J. Metz. The epidemic in a closed population with all susceptibles equally vulnerable; some results for large susceptible populations and small initial infections. *Acta Biotheoretica*, 27:75–123, 1978.
21. D. W. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.
22. D. W. Opitz and J. W. Shavlik. Actively searching for an effective neural-network ensemble. *Connection Science*, 8(3/4):337–353, 1996.
23. F. J. Provost and T. Fawcett. Robust classification systems for imprecise environments. In *Proceedings of the 16th National Conference on Artificial Intelligence*, pages 706–713, 1998.
24. J. R. Quinlan. Bagging, boosting, and c4.5. In *Proceedings of the 14th National Conference on Artificial Intelligence*, volume 1, pages 725–730, 1996.
25. J. R. Quinlan. Boosting first-order learning. *Algorithmic Learning Theory, 7th International Workshop, Lecture Notes in Computer Science*, 1160:143–155, 1996.
26. V. Santos Costa, A. Srinivasan, and R. Camacho. A note on two simple transformations for improving the efficiency of an ILP system. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 225–242. Springer-Verlag, 2000.
27. M. Sebag and C. Rouveirol. Tractable induction and classification in first-order logic via stochastic matching. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 888–893. Morgan Kaufmann, 1997.
28. A. Srinivasan. A study of two sampling methods for analysing large datasets with ILP. *Data Mining and Knowledge Discovery*, 3(1):95–123, 1999.
29. A. Srinivasan. *The Aleph Manual*, 2001.
30. A. Srinivasan, R. King, S. Muggleton, and M. Sternberg. Carcinogenesis predictions using ILP. In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Artificial Intelligence*, pages 273–287. Springer-Verlag, 1997.
31. J. Struyf and H. Blockeel. Efficient cross-validation in ILP. In Céline Rouveirol and Michèle Sebag, editors, *Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *Lecture Notes in Artificial Intelligence*, pages 228–239. Springer-Verlag, September 2001.

32. F. Zelezny, A. Srinivasan, and D. Page. Lattice-search runtime distributions may be heavy-tailed. In *The Twelfth International Conference on Inductive Logic Programming*. Springer Verlag, July 2002.
33. J. Zelle and R. Mooney. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 817–822, Washington, D.C., July 1993. AAAI Press/MIT Press.
34. S. Zemke. Bagging imperfect predictors. In *Proceedings of the International Conference on Artificial Neural Networks in Engineering, St. Louis, MI, USA*. ASME Press, 1999.
35. M. Zweig and G. Campbell. Receiver-operative characteristic. *Clinical Chemistry*, 39:561–577, 1993.

Kernels for Structured Data

Thomas Gärtner[1,3], John W. Lloyd[2], and Peter A. Flach[3]

[1] Knowledge Discovery, Fraunhofer Institut Autonome Intelligente Systeme, Germany
Thomas.Gaertner@ais.fraunhofer.de
[2] Computer Sciences Laboratory, Research School of Information Sciences and Engineering, The Australian National University
jwl@csl.anu.edu.au
[3] Machine Learning, Department of Computer Science, University of Bristol, UK
Peter.Flach@bristol.ac.uk

Abstract. Learning from structured data is becoming increasingly important. However, most prior work on kernel methods has focused on learning from attribute-value data. Only recently have researchers started investigating kernels for structured data. This paper describes how kernel definitions can be simplified by identifying the structure of the data and how kernels can be defined on this structure. We propose a kernel for structured data, prove that it is positive definite, and show how it can be adapted in practical applications.

1 Introduction

Support vector machines and other kernel methods [3, 15] have successfully been applied to various tasks in attribute-value learning. Much 'real-world' data, however, is structured – it has no natural representation as a tuple of constants. Defining kernels on individuals that cannot easily be described by a feature vector means crossing the boundary between attribute-value and relational learning. It enables support vector machines and other kernel methods to be applied more easily to complex representation spaces.

The *kernel trick* is to replace the inner product in the representation space by an inner product in some feature space. The definition of the inner product thus determines the hypothesis language. The same trick can also be used with representation spaces that do not have a natural inner product defined on them. By defining a 'valid' kernel on these representations, they can be embedded into some linear space.

Crucial to the success of kernel-based learning algorithms is the extent to which the semantics of the domain are reflected in the definition of the kernel. A 'good' kernel calculates a high similarity for examples in the same class and low similarity for examples in different classes. To express the semantics of the data in a machine-readable form, often strongly typed syntaxes are used. Syntax-driven kernels are an attempt to define 'good' kernels based on the semantics of the domain as described by the syntax of the representation. The definition of a kernel on structured data and the proof that this kernel is 'valid' are the main contributions of this paper.

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 66–83, 2003.

This kernel is to be seen as the default kernel for structured data in the same sense in which the canonical dot product can be seen as the default kernel for vectors of numbers. Such default kernels may not always be the best choice. For that reason, gaussian, polynomial, or normalised versions of default kernels can be used.

The outline of the paper is as follows. Section 2 introduces kernel methods and defines what is meant by 'valid' and 'good' kernels. Section 3 gives an account of our knowledge representation formalism, which is a typed higher-order logic. Section 4 defines a kernel on the terms of this logic. Section 5 describes how these kernels can be adapted to particular domains. Section 6 illustrates the application of this kernel by some examples. Finally, some concluding remarks are given.

2 Kernel Methods

We distinguish two components of kernel methods, the kernel machine and the kernel function. Different kernel machines tackle different learning tasks, e.g., support vector machines for supervised learning, support vector clustering [1] for unsupervised learning, and kernel principal component analysis [15] for feature extraction. The kernel machine encapsulates the learning task and the way in which a solution is looked for. The kernel function encapsulates the hypothesis language, i.e., how a solution is described.

2.1 Classes of Kernels

A useful distinction between different classes of kernels is based on 'driving-force'. We distinguish between semantics, syntax, model, and data as the driving-force of the kernel definition. A similar terminology has been used previously in the context of constructive induction algorithms [2].

Semantics is the ideal driving-force for the definition of proximities. It is related to so-called 'knowledge-driven' approaches of incorporating expert knowledge into the domain representation. *Syntax* is often used in typed systems to formally describe the semantics of the data. It is the most common driving force. In the simplest case, i.e., untyped attribute-value representation, it treats every attribute in the same way. *Models* extract useful knowledge from previous learning attempts. While this is often done to learn the semantics of the data from the data itself, it violates the encapsulation of the search strategy for kernel methods. *Data-driven* approaches use results obtained by analysing the training data. This also violates the encapsulation of the search strategy, as some search is needed for analysing the data.

As we want to maintain the modularity aspect, we will focus on syntax-driven approaches throughout the remainder of this paper, where syntax is understood as being carefully engineered to reflect the underlying semantics of the data.

2.2 Valid Kernels

Technically, a kernel k calculates an inner product in some feature space which is, in general, different from the representation space of the instances. The compu-

tational attractiveness of kernel methods comes from the fact that quite often a closed form of these 'feature space inner products' exists. Instead of performing the expensive transformation step ϕ explicitly, a kernel $k(x,y) = \langle \phi(x), \phi(y) \rangle$ calculates the inner product directly and performs the feature transformation only implicitly.

Whether, for a given function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, a feature transformation $\phi : \mathcal{X} \to \mathcal{H}$ into some Hilbert space[1] $\mathcal{H}$ exists, such that k is the inner product in $\mathcal{H}$ ($\mathcal{H} \supseteq span(\{\phi(x) \,|\, x \in \mathcal{X}\})$), can be checked by verifying that the function is positive definite. This means that any set, whether a linear space or not, that admits a positive definite kernel can be embedded into a linear space. Thus, throughout the paper, we take 'valid' to mean 'positive definite'. Here then is the definition of a positive definite kernel. ($\mathbb{Z}^+$ is the set of positive integers.)

Definition 1. *Let $\mathcal{X}$ be a set. A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a* positive definite kernel *on $\mathcal{X}$ if, for all $n \in \mathbb{Z}^+, x_1, \ldots, x_n \in \mathcal{X}$, and $c_1, \ldots, c_n \in \mathbb{R}$, it follows that $\sum_{i,j \in \{1,\ldots,n\}} c_i \, c_j \, k(x_i, x_j) \geq 0$.*

While it is not always easy to prove positive definiteness for a given kernel, positive definite kernels do have some nice closure properties. In particular, they are closed under sum, direct sum, multiplication by a scalar, product, tensor product, zero extension, pointwise limits, and exponentiation [5, 9].

2.3 Good Kernels

For a kernel method to perform well on some domain, validity of the kernel is not the only issue. While there is always a valid kernel that performs poorly ($k_0(x,y) = 0$), there is also always a valid kernel ($k_\nu(x,y) = \nu(x)\nu(y)$, where $\nu(z)$ is $+1$ if z is is a member of the concept and -1 otherwise) that performs ideally. We distinguish the following three issues crucial to 'good' kernels: completeness, correctness, and appropriateness. A similar terminology has been used previously in the context of constructive induction algorithms [2].

Completeness refers to the extent to which the knowledge incorporated in the proximity is sufficient for solving the problem at hand. A proximity is said to be complete if it takes into account all the information necessary to represent the concept that underlies the problem domain. *Correctness* refers to the extent to which the underlying semantics of the problem are obeyed in the proximity. *Appropriateness* refers to the extent to which examples that are close to each other in class membership are also 'close' to each other in the proximity space. Another frequently used term is 'smoothness of the kernel with respect to the class membership'.

Empirically, a correct and appropriate kernel exhibits two properties. A correct kernel separates the concept well, i.e., a learning algorithm achieves high accuracy when learning and validating on the same part of the data. An appropriate kernel generalises well, i.e., a learning algorithm achieves high accuracy when learning and validating on different parts of the data.

[1] A Hilbert space is a linear space endowed with a dot product and complete in the norm corresponding to that dot product.

3 Knowledge Representation

For a syntax-driven kernel definition, one needs a knowledge representation formalism that is able to accurately and naturally model the underlying semantics of the data. The knowledge representation formalism we use is based on the principles of using a typed syntax and representing individuals as (closed) terms. The theory behind this knowledge representation formalism can be found in [12] and a brief outline is given in this section. The typed syntax is important for pruning search spaces and for modelling as closely as possible the semantics of the data in a human- and machine-readable form. The individuals-as-terms representation is a natural generalisation of the attribute-value representation and collects all information about an individual in a single term.

The setting is a typed, higher-order logic that provides a variety of important data types, including sets, multisets, and graphs for representing individuals. The logic is based on Church's simple theory of types [4] with several extensions. First, we assume there is given a set of type constructors $\mathfrak{T}$ of various arities. Included in $\mathfrak{T}$ is the constructor Ω of arity 0. The domain corresponding to Ω is the set containing just *True* and *False*, that is, the boolean values. The *types* of the logic are built up from the set of type constructors and a set of parameters (that is, type variables), using the symbol $\rightarrow$ (for function types) and $\times$ (for product types). For example, there is a type constructor *List* used to provide the list types. Thus, if α is a type, then *List* α is the type of lists whose elements have type α. A closed type is a type not containing any parameters, the set of all closed types is denoted by $\mathfrak{S}^c$. Standard types include *Nat* (the type of natural numbers).

There is also a set $\mathfrak{C}$ of constants of various types. Included in $\mathfrak{C}$ are $\top$ (true) and $\bot$ (false). Two different kinds of constants, *data constructors* and *functions*, are distinguished. In a knowledge representation context, data constructors are used to represent individuals. In a programming language context, data constructors are used to construct data values. (Data constructors are called functors in Prolog.) In contrast, functions are used to compute on data values; functions have definitions while data constructors do not. In the semantics for the logic, the data constructors are used to construct models (cf. Herbrand models for Prolog). A *signature* is the declared type of a constant. For example, the empty list constructor [] has signature *List* a, where a is a parameter. The list constructor : (usually written infix) has signature $a \rightarrow \mathit{List}\ a \rightarrow \mathit{List}\ a$ [2]. Thus : expects two arguments, an element of type a and a list of type *List* a, and produces a new list of type *List* a. If a constant C has signature α, we denote this by $C : \alpha$.

The *terms* of the logic are the terms of the typed λ-calculus, which are formed in the usual way by abstraction, tupling, and application from constants in $\mathfrak{C}$ and a set of variables. $\mathfrak{L}$ denotes the set of all terms (obtained from a particular alphabet). A term of type Ω is called a *formula*. A function whose codomain type is Ω is called a *predicate*. In the logic, one can introduce the usual connectives and quantifiers as functions of appropriate types. Thus the connectives conjunction, $\wedge$, and disjunction, $\vee$, are functions of type $\Omega \rightarrow \Omega \rightarrow \Omega$. In addition, if t is of

[2] This could be read as $a \times \mathit{List}\ a \rightarrow \mathit{List}\ a$.

type Ω, the abstraction $\lambda x.t$ is written $\{x \mid t\}$ to emphasise its intended meaning as a set. There is also a tuple-forming notation $(\ldots)$. Thus, if $t_1, \ldots, t_n$ are terms of type $\tau_1, \ldots, \tau_n$, respectively, then $(t_1, \ldots, t_n)$ is a term of type $\tau_1 \times \cdots \times \tau_n$.

Now we come to the key definition of basic terms. Intuitively, basic terms represent the individuals that are the subject of learning (in Prolog, these would be the ground terms). Basic terms fall into one of three kinds: those that represent individuals that are lists, trees, and so on; those that represent sets, multisets, and so on; and those that represent tuples. The second kind are abstractions. For example, the basic term representing the set $\{1, 2\}$ is

$$\lambda x.\mathit{if}\ x = 1\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = 2\ \mathit{then}\ \top\ \mathit{else}\ \bot,$$

and

$$\lambda x.\mathit{if}\ x = A\ \mathit{then}\ 42\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ 0$$

is the representation of the multiset with 42 occurrences of A and 21 occurrences of B (and nothing else). Thus we adopt abstractions of the form

$$\lambda x.\mathit{if}\ x = t_1\ \mathit{then}\ s_1\ \mathit{else}\ \ldots\ \mathit{if}\ x = t_n\ \mathit{then}\ s_n\ \mathit{else}\ s_0$$

to represent (extensional) sets, multisets, and so on. The term s_0 here is called a default term and for the case of sets is $\bot$ and for multisets is 0. Generally, one can define default terms for each (closed) type. The set of default terms is denoted by $\mathfrak{D}$. (Full details on default terms are given in [12].)

Definition 2. *The set of* basic terms, $\mathfrak{B}$, *is defined inductively as follows.*

1. *If* C *is a data constructor having signature* $\sigma_1 \to \cdots \to \sigma_n \to (T\ a_1 \ldots a_k)$, $t_1, \ldots, t_n \in \mathfrak{B}$ $(n \geq 0)$, *and* t *is* $C\ t_1 \ldots t_n \in \mathfrak{L}$, *then* $t \in \mathfrak{B}$.
2. *If* $t_1, \ldots, t_n \in \mathfrak{B}$, $s_1, \ldots, s_n \in \mathfrak{B}$ $(n \geq 0)$, $s_0 \in \mathfrak{D}$ *and* t *is*

$$\lambda x.\mathit{if}\ x = t_1\ \mathit{then}\ s_1\ \mathit{else}\ \ldots\ \mathit{if}\ x = t_n\ \mathit{then}\ s_n\ \mathit{else}\ s_0 \in \mathfrak{L},$$

   then $t \in \mathfrak{B}$.
3. *If* $t_1, \ldots, t_n \in \mathfrak{B}$ $(n \geq 0)$ *and* t *is* $(t_1, \ldots, t_n) \in \mathfrak{L}$, *then* $t \in \mathfrak{B}$.

Part 1 of the definition of the set of basic terms states, in particular, that individual natural numbers, integers, and so on, are basic terms. Also a term formed by applying a data constructor to (all of) its arguments, each of which is a basic term, is a basic term. For example, lists are formed using the data constructors [] having signature $\mathit{List}\ a$, and : having signature $a \to \mathit{List}\ a \to \mathit{List}\ a$. Thus $A : B : C : []$ is the basic term of type $\mathit{List}\ \alpha$ representing the list $[A, B, C]$, where A, B, and C are constants having signature α. Basic terms coming from Part 1 of the definition are called *basic structures* and always have a type of the form $T\alpha_1 \ldots \alpha_n$.

The abstractions formed in Part 2 of the definition are "almost constant" abstractions since they take the default term s_0 as value for all except a finite

number of points in the domain. They are called *basic abstractions* and always have a type of the form $\beta \to \gamma$. This class of abstractions includes useful data types such as (finite) sets and multisets (assuming $\perp$ and 0 are default terms). More generally, basic abstractions can be regarded as lookup tables, with s_0 as the value for items not in the table. In fact, the precise definition of basic terms in [12] is a little more complicated in that, in the definition of basic abstractions, $t_1, \ldots, t_n$ are ordered and $s_1, \ldots, s_n$ cannot be default terms. These conditions avoid redundant representations of abstractions.

Part 3 of the definition of basic terms just states that one can form a tuple from basic terms and obtain a basic term. These terms are called *basic tuples* and always have a type of the form $\alpha_1 \times \cdots \times \alpha_n$.

Compared with Prolog, our knowledge representation offers a type system which can be used to express the structure of the hypothesis space and thus acts as a declarative bias. The other important extension are the abstractions, which allow us to use genuine sets and multisets. In fact, Prolog only has data constructors (functors), which are also used to emulate tuples.

It will be convenient to gather together all basic terms that have a type more general than some specific closed type. In this definition, if α and β are types, then α is *more general than* β if there exists a type substitution ξ such that $\beta = \alpha\xi$.

Definition 3. *For each* $\alpha \in \mathfrak{S}^c$, *define* $\mathfrak{B}_\alpha = \{t \in \mathfrak{B} \mid t$ *has type more general than* $\alpha\}$.

The intuitive meaning of $\mathfrak{B}_\alpha$ is that it is the set of terms representing individuals of type α.

For use in the definition of a kernel, we introduce some notation. If $s \in \mathfrak{B}_{\beta\to\gamma}$ and $t \in \mathfrak{B}_\beta$, then $V(s\ t)$ denotes the "value" returned when s is applied to t. (The precise definition is in [12].) For example, if s is $\lambda x.$*if* $x = A$ *then* 42 *else if* $x = B$ *then* 21 *else* 0 and t is A, then $V(s\ t) = 42$. Also, if $u \in \mathfrak{B}_{\beta\to\gamma}$, the *support* of u, denoted $supp(u)$, is the set $\{v \in \mathfrak{B}_\beta \mid V(u\ v) \notin \mathfrak{D}\}$. Thus, for the s above, $supp(s) = \{A, B\}$.

As an example of the use of the formalism, for (directed) graphs, there is a type constructor *Graph* such that the type of a graph is *Graph* $\nu\ \varepsilon$, where ν is the type of information in the vertices and ε is the type of information in the edges. *Graph* is defined by

$$Graph\ \nu\ \varepsilon = \{Label \times \nu\} \times \{(Label \times Label) \times \varepsilon\},$$

where *Label* is the type of labels. Note that this definition corresponds closely to the mathematical definition of a graph: each vertex is uniquely labelled and each edge is uniquely labelled by the ordered pair of labels of the vertices it connects.

4 Embedding Basic Terms in Linear Spaces

Having introduced kernels (in Section 2) and our knowledge representation formalism (in Section 3), we are now ready to define default kernels for basic terms.

This definition of a kernel on basic terms assumes the existence of kernels on the various sets of data constructors. More precisely, for each type constructor $T \in \mathfrak{T}$, κ_T is assumed to be a positive definite kernel on the set of data constructors associated with T. For example, for the type constructor Nat, κ_{Nat} could be the *product kernel* defined by $\kappa_{Nat}(m,n) = mn$. For a type constructor M, the *matching kernel* κ_M is defined by $\kappa_M(x,y) = k_\delta(x,y) = 1$ if $x = y$, and 0, otherwise.

Definition 4. *The function $k : \mathfrak{B} \times \mathfrak{B} \to \mathbb{R}$ is defined inductively on the structure of terms in $\mathfrak{B}$ as follows. Let $s, t \in \mathfrak{B}$.*

1. *If $s, t \in \mathfrak{B}_\alpha$, where $\alpha = T\ \alpha_1 \ldots \alpha_k$, for some $T, \alpha_1, \ldots, \alpha_k$, then*

$$k(s,t) = \begin{cases} \kappa_T(C,D) & \text{if } C \neq D \\ \kappa_T(C,C) + \sum_{i=1}^{n} k(s_i,t_i) & \text{otherwise} \end{cases}$$

   where s is $C\ s_1 \ldots s_n$ and t is $D\ t_1 \ldots t_m$.
2. *If $s, t \in \mathfrak{B}_\alpha$, where $\alpha = \beta \to \gamma$, for some β, γ, then*

$$k(s,t) = \sum_{\substack{u \in supp(s) \\ v \in supp(t)}} k(V(s\ u), V(t\ v)) \cdot k(u,v).$$

3. *If $s, t \in \mathfrak{B}_\alpha$, where $\alpha = \alpha_1 \times \cdots \times \alpha_n$, for some $\alpha_1, \ldots, \alpha_n$, then*

$$k(s,t) = \sum_{i=1}^{n} k(s_i,t_i),$$

   where s is $(s_1, \ldots, s_n)$ and t is $(t_1, \ldots, t_n)$.
4. *If there does not exist $\alpha \in \mathfrak{S}^c$ such that $s, t \in \mathfrak{B}_\alpha$, then $k(s,t) = 0$.*

Example 1. Suppose that κ_{List} is the matching kernel. Let M be a nullary type constructor and $A, B, C, D : M$. Suppose that κ_M is the matching kernel. Let s be the list $[A, B, C]$ and t the list $[A, D]$. Then

$$\begin{aligned} k(s,t) &= \kappa_{List}((:),(:)) + k(A,A) + k([B,C],[D]) \\ &= 1 + \kappa_M(A,A) + \kappa_{List}((:),(:)) + k(B,D) + k([C],[]) \\ &= 1 + 1 + 1 + \kappa_M(B,D) + \kappa_{List}((:),[]) \\ &= 3 + 0 + 0 \\ &= 3. \end{aligned}$$

Thus, the kernel counts 1 for the first list constructor in both terms, 1 for the matching heads of the list, and 1 for the second list constructor, after which the two terms differ. Notice that the recursive matching of the two lists as performed by the kernel is similar to the kind of matching that is performed in anti-unification.

Example 2. Suppose that κ_Ω is the matching kernel. Let M be a nullary type constructor and $A, B, C, D : M$. Suppose that κ_M is the matching kernel. If s is the set $\{A, B, C\} \in \mathfrak{B}_{M\to\Omega}$ and t is the set $\{A, D\} \in \mathfrak{B}_{M\to\Omega}$, then

$$\begin{aligned} k(s,t) &= k(A,A) + k(A,D) + k(B,A) + k(B,D) + k(C,A) + k(C,D) \\ &= \kappa_M(A,A) + \kappa_M(A,D) + \kappa_M(B,A) + \kappa_M(B,D) + \kappa_M(C,A) \\ &\quad + \kappa_M(C,D) \\ &= 1 + 0 + 0 + 0 + 0 + 0 \\ &= 1. \end{aligned}$$

Thus, the kernel performs a pairwise match of the elements of each set. This could equivalently be seen as the inner product of the bitvectors representing the two sets.

Example 3. Suppose that κ_{Nat} is the product kernel. Let M be a nullary type constructor and $A, B, C, D : M$. Suppose that κ_M is the matching kernel. If s is $\langle A, A, B, C, C, C\rangle \in \mathfrak{B}_{M\to Nat}$ (where $\langle A, A, B, C, C, C\rangle$ is the multiset containing two occurrences of A, one of B, and three of C) and t is $\langle B, C, C, D\rangle \in \mathfrak{B}_{M\to Nat}$, then

$$\begin{aligned} k(s,t) &= k(2,1)k(A,B) + k(2,2)k(A,C) + k(2,1)k(A,D) \\ &\quad + k(1,1)k(B,B) + k(1,2)k(B,C) + k(1,1)k(B,D) \\ &\quad + k(3,1)k(C,B) + k(3,2)k(C,C) + k(3,1)k(C,D) \\ &= 1 \times 1 + 6 \times 1 \\ &= 7. \end{aligned}$$

If we represent multisets by multiplicity vectors, we again have that the kernel computes their inner product.

We can now formulate the main theoretical result of the paper.

Proposition 1. *Let $k : \mathfrak{B} \times \mathfrak{B} \to \mathbb{R}$ be the function defined in Definition 4. For each $\alpha \in \mathfrak{S}^c$, k is a positive definite kernel on $\mathfrak{B}_\alpha$.*

The inductive proof of Proposition 1 is given in the appendix. Here is an intuitive outline. First, assume that those kernels occurring on the right-hand side of each kernel definition are positive definite. Then the positive definiteness of the (left-hand side) kernel follows from the closure properties of the class of positive definite kernels. The kernel on basic structures is positive definite because of closure under sum, zero extension, and direct sum, and because the kernels defined on the data constructors are assumed to be positive definite. The kernel on basic abstractions is positive definite as the function *supp* returns a finite set, and kernels are closed under zero extension, sum, and tensor product. The kernel on basic tuples is positive definite because of closure under direct sum.

5 Adapting Kernels

The kernel defined in the previous section closely follows the type structure of the individuals that are used for learning. As indicated, the kernel assumes the existence of atomic kernels for all data constructors used. These kernels can be the product kernel for numbers, the matching kernel which just checks whether the two constructors are the same, or a user-defined kernel. In addition, kernel modifiers can be used to customise the kernel definition to the domain at hand. In this section we first describe some commonly used kernel modifiers. After that, we suggest how atomic kernels and kernel modifiers can be specified by an extension of the Haskell language [11].

To incorporate domain knowledge into the kernel definition, it will frequently be necessary to modify the default kernels for a type. Below we formally describe these modifications in terms of a function $\kappa_{\text{modifier}} : \mathcal{P} \to (\mathcal{X} \times \mathcal{X} \to \mathbb{R}) \to (\mathcal{X} \times \mathcal{X} \to \mathbb{R})$ that – given a modifier and its parameters (an element of the parameter space $\mathcal{P}$) – maps any kernel to the modified kernel. For these modifiers, several choices are offered.

By default, no modifier is used, i.e.,

$$\kappa_{\text{default}}(k)(x,y) = k(x,y).$$

Instead, a polynomial version of the default kernel can be used:

$$\kappa_{\text{polynomial}}(p,l)(k)(x,y) = (k(x,y)+l)^p\,. \qquad (l \geq 0,\ p \in \mathbb{Z}^+)$$

Or a gaussian version:

$$\kappa_{\text{gaussian}}(\gamma)(k)(x,y) = e^{-\gamma[k(x,x)-2k(x,y)+k(y,y)]}. \qquad (\gamma > 0)$$

Another frequently used modification is the normalisation kernel:

$$\kappa_{\text{normalised}}(k)(x,y) = \frac{k(x,y)}{\sqrt{k(x,x)k(y,y)}}.$$

Other modifiers can be defined by the user.

We suggest that kernels be defined directly on the type structure (specifying the structure of the domain and the declarative bias). We introduce our suggested kernel definition syntax by means of an example: the East/West challenge [13].

```
eastbound :: Train -> Bool
type Train = Car -> Bool with modifier (gaussian 0.1)
type Car = (Shape,Length,Roof,Wheels,Load)
data Shape = Rectangle | Oval
data Length = Long | Short
data Roof = Flat | Peaked | None with kernel roofKernel
type Wheels = Int with kernel discreteKernel
type Load = (LShape,LNumber)
data LShape = Rectangle | Circle | Triangle
type LNumber = Int
```

The first line declares the learning target `eastbound` as a mapping from trains to Booleans. A train is a set of cars, and a car is a 5-tuple describing its shape, its length, its roof, its number of wheels, and its load. All of these are specified by data constructors except the load, which itself is a pair of data constructors describing the shape and number of loads.

The `with` keyword describes a property of a type, in this case kernels and kernel modifiers. The above declarations state that on trains we use a Gaussian kernel modifier with bandwidth 0.1. By default, for `Shape`, `Length` and `LShape` the matching kernel is used, while for `LNumber` the product kernel is used. The default kernel is overridden for `Wheels`, which is defined as an integer but uses the matching kernel instead. Finally, `Roof` has been endowed with a user-defined atomic kernel which could be defined as follows:

```
roofKernel :: Roof -> Roof -> Real
roofKernel x x = 1
roofKernel Flat Peaked = 0.5
roofKernel Peaked Flat = 0.5
roofKernel x y = 0
```

This kernel counts 1 for identical roofs, 0.5 for matching flat against peaked roofs, and 0 in all other cases (i.e., whenever one car is open and the other is closed).

Finally, the normalisation modifier could be implemented as follows:

```
normalised :: (t->t->Real) -> t -> t -> Real
normalised k x y = (k x y) / sqrt ((k x x) * (k y y))
```

6 Example Applications

Having presented our kernel definition, it is important to note that aside from being used with kernel methods, our kernel function can also be used with other (dis-)similarity-based algorithms. A normalised kernel is a similarity function; a metric can be defined from the kernel in the standard manner ($d_k(x, y) = \sqrt{k(x, x) - 2k(x, y) + k(y, y)}$). Thus learning algorithms like nearest neighbour and k-means can easily be extended to structured data.

In this section, we empirically investigate the appropriateness of our kernel definitions on some domains. The implementation of most algorithms mentioned below has been simplified by using the Weka data mining toolkit [16].

6.1 East/West Challenge

We performed some experiments with the East/West challenge dataset. We used the default kernels for all types, i.e., the product kernel for all numbers, the matching kernel for all other atomic types, and no kernel modifiers. As this toy data set only contains 20 labelled instances, the aim of our experiments was not to achieve a high predictive accuracy but to check whether this problem

can actually be separated using our default kernel. For that, we applied a support vector machine and a 3-nearest-neighbour classifier to the full data set. In both experiments, we achieved 100% accuracy, verifying that the data is indeed separable with the default kernels.

6.2 Spatial Clustering

Consider the problem of clustering spatially close and thematically similar data points. This problem occurs, for example, when given demographic data about households in a city and trying to optimise facility locations given this demographic data. The location planning algorithms can usually only deal with a fairly small number of customers (less than 1000) and even for small cities the number of households easily exceeds 10000. Therefore, several households have to be aggregated so that as little information as possible is lost. Thus the households that are aggregated have to be spatially close (so that little geographic information is lost) and similar in their demographic description (so that little demographic information is lost). The problem is to automatically find such an aggregation using an unsupervised learning algorithm.

Due to the difficulty in obtaining suitable data, we investigated this problem on a slightly smaller scale. The demographic data was already aggregated for data protection and anonymity reasons such that information is given not on a household level but on a (part of) street level. The data set describes roughly 500 points in a small German city by its geographic co-ordinates and 76 statistics, e.g., the number of people above or below certain age levels, the number of people above or below certain income levels, and the number of males or females living in a small area around the data point.

The simplest way to represent this data is a feature vector with 78 entries (2 for the x, y co-ordinates and 76 for the statistics). Drawing the results of a simple k-means algorithm on this representation clearly shows that although the spatial co-ordinates are taken into account, spatially compact clusters cannot be achieved. This is due to the fact that the semantics of the co-ordinates and the demographic statistics are different.

An alternative representation along with the kernel specification is as follows:

```
type SpacialStat = GeoInfo -> Statistics
type GeoInfo = (Real,Real) with modifier (gaussian 0.1)
type Statistics = (Real,Real,...,Real) with modifier normalised
```

Using this representation and applying a version of the k-means algorithm[3] with the given kernel shows that the clusters are spatially compact (compactness depending on the choice of the kernel bandwidth).

Illustrations of results can be found on-line[4]. Instances belonging to the same cluster are represented by the same colour. The street map and the buildings of the city are shown in grey.

[3] Note that performing k-means in feature space requires some modifications of the algorithm. A description is beyond the scope of this paper.

[4] `http://www.ais.fraunhofer.de/~thomasg/SpatialClustering/`

6.3 Drug Activity Prediction

A frequently used concept class on data with limited structure are multi-instance concepts. Multi-instance learning problems occur whenever individuals cannot be described by a single characteristic feature vector, but require a bag of vectors. A popular real-world example of a multi-instance problem is the prediction of drug activity, introduced in [6].

It is common in drug activity prediction to represent a molecule by a bag of descriptions of its different conformations. A drug is active if one of its conformations binds well to enzymes or cell-surface receptors. Each conformation is described by a feature vector where each component corresponds to one ray emanating from the origin and measuring the distance to the molecule surface. In the musk domain, 162 uniformly distributed rays have been chosen to represent each conformation. Additionally, four further features are used that describe the position of an oxygen atom in the conformation.

The formal specification of the structure of the musk data set along with the kernel applied in [8] is as follows:

```
type Molecule = Con -> Int
type Con = (Rays,Distance,Offset) with modifier (gaussian 1e-5.5)
type Rays = (Real,Real,...,Real)
type Offset = (Real,Real,Real)
type Distance = Real
```

The best result achieved in the literature on musk1 is 96.8%, and the next five best results range between 92.4% and 88.9%. The support vector machine using the above kernel achieved 87% accuracy, better results can be achieved with smaller γ. The best result achieved in the literature on musk2 is 96.0%, and the next five best results range between 89.2% and 82.5%. The support vector machine using the above kernel achieved 92.2% accuracy. For a more detailed evaluation and discussion of these results, the reader is referred to [8].

7 Related Work

7.1 Kernels on Discrete Structures

Below we summarize prior work on syntax-driven kernels for discrete spaces that is most relevant in our context.

The best known kernel for representation spaces that are not mere attribute-value tuples is the convolution kernel proposed by Haussler [9]. It is defined there as

$$k_{conv}(x,y) = \sum_{\boldsymbol{x}\in R^{-1}(x),\boldsymbol{y}\in R^{-1}(y)} \prod_{d=1}^{D} k_d(x_d, y_d),$$

where R is a relation between instances x and their parts, i.e., R^{-1} decomposes an instance into a set of D-tuples. The term 'convolution kernel' refers to a

class of kernels that can be formulated in the above way. The advantage of convolution kernels is that they are very general and can be applied in many different problems. However, because of that generality, they require a significant amount of work to adapt them to a specific problem, which makes choosing R a non-trivial task.

More specific kernels on discrete spaces have been described in [7]. There, kernels for elementary symbols, sets and multi-sets of elementary symbols, and Boolean domains are discussed along with concept classes that can be separated by linear classifiers using these kernels. An overview along with various extensions can be found in [8].

7.2 Distances on Discrete Structures

For all kernels $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ we can define a valid metric[5] by

$$d_k(x, y) = \sqrt{k(x, x) - 2k(x, y) + k(y, y)}$$

The opposite, however, does not necessarily hold. Given a distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ we can define a function k_d from the distance by

$$k_d(x, y) = \frac{1}{2}\left[||x||^2 - d^2(x, y) + ||y||^2\right]$$

However, this function is not necessarily a valid kernel.

For Euclidean vector spaces $\mathcal{X} = \mathbb{R}^n$ it is easy to see that with any Euclidean distance $d(x, y) = \sqrt{\sum_i^n (x_i - y_i)^2}$, k_d is a positive definite kernel (by using the Euclidean distance a Euclidean vector space becomes a Hilbert space). Obviously, k_d is also a positive definite kernel, if the function d corresponds to a Euclidean distance in some feature space. Conversely, if no such feature transformation exists k_d is not positive definite. Notice that it can easily be shown that if d is not a pseudo-metric[6], then it does not correspond to a Euclidean distance in any feature space, and therefore k_d is not a positive definite kernel.

Distance functions on discrete structures have been investigated for some time. However, for most distance functions defined in literature the metric properties do not hold. For example, one well know distance function on discrete structures is the RIBL distance, described in [10], for which neither symmetry nor the triangle inequality hold. Several other distances are summarised in [10, 14] where it is shown that most distances suggested so far are either non-metric or unsuited for real-world problems. An exeption to this is [14]. The distance function suggested there is based on the idea of computing a legal flow of maximal flow value and minimal cost in a weighted bipartite graph. It can be shown that this distance satisfies all properties of a metric. However, to the best of our knowledge, it has not yet been shown whether this distance is an Euclidean distance in some space or not.

[5] A metric satisfies the triangle inequality $d(x, y) + d(y, z) \geq d(x, z)$, is symmetric $d(x, y) = d(y, x)$, and satisfies $d(x, y) = 0 \Leftrightarrow x = y$.

[6] For a pseudo-metric the distance between different points can be zero, i.e., the last of the above metric properties is replaced by $d(x, y) = 0 \Leftarrow x = y$.

8 Conclusions

Bringing together kernel methods and structured data is an important direction for practical machine learning research. This can be done by defining a positive definite kernel on structured data and thus embedding structured data into a linear space. In this paper we defined a kernel on structured data, proved that it is positive definite, and showed some example applications.

Our kernel definition follows a 'syntax-driven' approach making use of a knowledge representation formalism that is able to accurately and naturally model the underlying semantics of structured data. It is based on the principles of using a typed syntax and representing individuals as (closed) terms. The typed syntax is important for pruning search spaces and for modelling as closely as possible the semantics of the data in a human- and machine-readable form. The individuals-as-terms representation is a simple and natural generalisation of the attribute-value representation and collects all information about an individual in a single term. In spite of this simplicity, the knowledge representation formalism is still powerful enough to accurately model highly structured data such as graphs.

The definition of our kernel, along with the example applications presented above, show that structured data can reasonably be embedded in linear spaces. The embedding is complete in the sense that it incorporates all information that is present in an individual. Its correctness has been illustrated on a toy example of classifying trains; its appropriateness has been verified on a drug activity prediction domain.

Future work will consider both extending and specialising the kernel function presented above. Extending the kernel definition means, for example, considering abstractions that are not constant for a set of finite measure rather than being not constant only for a finite set. Specialising the framework implies the definition of specialised kernel functions for some type constructors that have special semantics. Future work will also further evaluate the applicability of the above presented kernel definition on a wide range of real world applications.

Acknowledgements

Research supported in part by the Esprit V project (IST-1999-11495) *Data Mining and Decision Support for Business Competitiveness: Solomon Virtual Enterprise* and by the BMBF funded project *KogiPlan*.

References

1. A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik. Support vector clustering. *Journal of Machine Learning Research*, 2:125–137, Dec. 2001.
2. E. Bloedorn, R. Michalski, and J. Wnek. Matching methods with problems: A comparative analysis of constructive induction approaches. Technical report, Machine Learning and Inference Laboratory, MLI 94-2, George Mason University, Fairfax, VA., 1994.

3. B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, July 1992. ACM Press.
4. A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
5. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines (and Other Kernel-Based Learning Methods).* Cambridge University Press, 2000.
6. T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1–2):31–71, 1997.
7. T. Gärtner. Kernel-based feature space transformation in inductive logic programming. Master's thesis, University of Bristol, 2000.
8. T. Gärtner, P. A. Flach, A. Kowalczyk, and A. J. Smola. Multi-instance kernels. In C. Sammut and A. Hoffmann, editors, *Proceedings of the 19th International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann, June 2002.
9. D. Haussler. Convolution kernels on discrete structures. Technical report, Department of Computer Science, University of California at Santa Cruz, 1999.
10. T. Horváth, S. Wrobel, and U. Bohnebeck. Relational instance-based learning with lists and terms. *Machine Learning*, 43(1/2):53–80, Apr. 2001.
11. S. P. Jones and J. Hughes, editors. *Haskell98: A Non-Strict Purely Functional Language.* Available at `http://haskell.org/`.
12. J. Lloyd. Knowledge representation, computation, and learning in higher-order logic. Available at `http://csl.anu.edu.au/~jwl`, 2001.
13. D. Michie, S. Muggleton, D. Page, and A. Srinivasan. To the international computing community: A new eastwest challenge. Technical report, Oxford University Computing laboratory, Oxford,UK, 1994.
14. J. Ramon and M. Bruynooghe. A polynomial time computable metric between point sets. *Acta Informatica*, 37(10):765–780, 2001.
15. B. Schölkopf and A. J. Smola. *Learning with Kernels.* The MIT Press, 2002.
16. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java implementations.* Morgan Kaufmann, 2000.

A Proof of the Proposition

Before giving the proof, which is an induction argument, some preparation is needed. The key idea is to base the induction on a 'bottom-up' definition of $\mathfrak{B}$. Here is the relevant definition.

Definition 5. *Define* $\{\mathfrak{B}_m\}_{m\in\mathbb{N}}$ *inductively as follows.*

$$\begin{aligned}
\mathfrak{B}_0 &= \{C \mid C \text{ is a data constructor of arity } 0\}\\
\mathfrak{B}_{m+1} &= \{C\ t_1 \ldots t_n \in \mathfrak{L} \mid C \text{ is a data constructor of arity } n \text{ and}\\
&\qquad\qquad t_1,\ldots,t_n \in \mathfrak{B}_m (n \geq 0)\}\\
&\cup \{\lambda x.\textit{if } x = t_1 \textit{ then } s_1 \textit{ else } \ldots \textit{ if } x = t_n \textit{ then } s_n \textit{ else } s_0 \in \mathfrak{L} \mid\\
&\qquad\qquad t_1,\ldots,t_n \in \mathfrak{B}_m, s_1,\ldots,s_n \in \mathfrak{B}_m, \textit{ and } s_0 \in \mathfrak{D}\}\\
&\cup \{(t_1,\ldots,t_n) \in \mathfrak{L} \mid t_1,\ldots,t_n \in \mathfrak{B}_m\}.
\end{aligned}$$

One can prove that $\mathfrak{B}_m \subseteq \mathfrak{B}_{m+1}$, for $m \in \mathbb{N}$, and that $\mathfrak{B} = \bigcup_{m\in\mathbb{N}} \mathfrak{B}_m$. Now the proof of Proposition 1 can be given.

Proof. First the symmetry of k on each $\mathfrak{B}_\alpha$ is established. For each $m \in \mathbb{N}$, let $SYM(m)$ be the property:

For all $\alpha \in \mathfrak{S}^c$ and $s, t \in \mathfrak{B}_\alpha \cap \mathfrak{B}_m$, it follows that $k(s,t) = k(t,s)$.

It is shown by induction that $SYM(m)$ holds, for all $m \in \mathbb{N}$. The symmetry of k on each $\mathfrak{B}_\alpha$ follows immediately from this since, given $s, t \in \mathfrak{B}_\alpha$, there exists an m such that $s, t \in \mathfrak{B}_m$ (because $\mathfrak{B} = \bigcup_{m\in\mathbb{N}} \mathfrak{B}_m$ and $\mathfrak{B}_m \subseteq \mathfrak{B}_{m+1}$, for all $m \in \mathbb{N}$).

First it is shown that $SYM(0)$ holds. In this case, s and t are data constructors of arity 0 associated with the same type constructor T, say. By definition, $k(s,t) = \kappa_T(s,t)$ and the result follows because κ_T is symmetric.

Now assume that $SYM(m)$ holds. It is proved that $SYM(m+1)$ also holds. Thus suppose that $\alpha \in \mathfrak{S}^c$ and $s, t \in \mathfrak{B}_\alpha \cap \mathfrak{B}_{m+1}$. It has to be shown that $k(s,t) = k(t,s)$. There are three cases to consider corresponding to α having the form $T\ \alpha_1 \ldots \alpha_k$, $\beta \to \gamma$, or $\alpha_1 \times \cdots \times \alpha_m$. In each case, it is easy to see from the definition of k and the induction hypothesis that $k(s,t) = k(t,s)$. This completes the proof that k is symmetric on each $\mathfrak{B}_\alpha$.

For the remaining part of the proof, for each $m \in \mathbb{N}$, let $PD(m)$ be the property:

For all $n \in \mathbb{Z}^+, \alpha \in \mathfrak{S}^c, t_1, \ldots, t_n \in \mathfrak{B}_\alpha \cap \mathfrak{B}_m$, and $c_1, \ldots, c_n \in \mathbb{R}$, it follows that $\sum_{i,j\in\{1,\ldots,n\}} c_i\ c_j\ k(t_i, t_j) \geq 0$.

It is shown by induction that $PD(m)$ holds, for all $m \in \mathbb{N}$. The remaining condition for positive definiteness follows immediately from this since, given $t_1, \ldots, t_n \in \mathfrak{B}_\alpha$, there exists an m such that $t_1, \ldots, t_n \in \mathfrak{B}_m$.

First it is shown that $PD(0)$ holds. In this case, each t_i is a data constructor of arity 0 associated with the same type constructor T, say. By definition, $k(t_i, t_j) = \kappa_T(t_i, t_j)$, for each i and j, and the result follows since κ_T is assumed to be positive definite.

Now assume that $PD(m)$ holds. It is proved that $PD(m+1)$ also holds. Thus suppose that $n \in \mathbb{Z}^+, \alpha \in \mathfrak{S}^c, t_1, \ldots, t_n \in \mathfrak{B}_\alpha \cap \mathfrak{B}_{m+1}$, and $c_1, \ldots, c_n \in \mathbb{R}$. It has to be shown that $\sum_{i,j\in\{1,\ldots,n\}} c_i\ c_j\ k(t_i, t_j) \geq 0$. There are three cases to consider.

1. Let $\alpha = T\ \alpha_1 \ldots \alpha_k$. Suppose that $t_i = C_i\ t_i^{(1)} \ldots t_i^{(m_i)}$, where $m_i \geq 0$, for $i = 1, \ldots, n$. Let $\mathcal{C} = \{C_i \mid i = 1, \ldots, n\}$. Then

$$\begin{aligned}
&\sum_{i,j\in\{1,\ldots,n\}} c_i\ c_j\ k(t_i, t_j) \\
&= \sum_{i,j\in\{1,\ldots,n\}} c_i\ c_j\ \kappa_T(C_i, C_j) \\
&\quad + \sum_{\substack{i,j\in\{1,\ldots,n\} \\ C_i = C_j}} c_i\ c_j \sum_{l\in\{1,\ldots,arity(C_i)\}} k(t_i^{(l)}, t_j^{(l)}).
\end{aligned}$$

Now

$$\sum_{i,j\in\{1,\ldots,n\}} c_i\ c_j\ \kappa_T(C_i,C_j) \geq 0$$

using the fact that κ_T is a positive definite kernel on the set of data constructors associated with T. Also

$$\begin{aligned}
&\sum_{\substack{i,j\in\{1,\ldots,n\}\\ C_i=C_j}} c_i\ c_j \sum_{l\in\{1,\ldots,arity(C_i)\}} k(t_i^{(l)},t_j^{(l)}) \\
&= \sum_{C\in\mathcal{C}} \sum_{\substack{i,j\in\{1,\ldots,n\}\\ C_i=C_j=C}} \sum_{l\in\{1,\ldots,arity(C)\}} c_i\ c_j\ k(t_i^{(l)},t_j^{(l)}) \\
&= \sum_{C\in\mathcal{C}} \sum_{l\in\{1,\ldots,arity(C)\}} \sum_{\substack{i,j\in\{1,\ldots,n\}\\ C_i=C_j=C}} c_i\ c_j\ k(t_i^{(l)},t_j^{(l)}) \\
&\geq 0,
\end{aligned}$$

by the induction hypothesis.

2. Let $\alpha = \beta \to \gamma$. Then

$$\begin{aligned}
&\sum_{i,j\in\{1,\ldots,n\}} c_i\ c_j\ k(t_i,t_j) \\
&= \sum_{i,j\in\{1,\ldots,n\}} c_i\ c_j \sum_{\substack{u\in supp(t_i)\\ v\in supp(t_j)}} k(V(t_i\ u),V(t_j\ v))\cdot k(u,v) \\
&= \sum_{i,j\in\{1,\ldots,n\}} \sum_{\substack{u\in supp(t_i)\\ v\in supp(t_j)}} c_i\ c_j\ k(V(t_i\ u),V(t_j\ v))\cdot k(u,v) \\
&= \sum_{\substack{(i,u),(j,v)\in\\ \{(k,w)\mid k=1,\ldots,n \text{ and } w\in supp(t_k)\}}} c_i\ c_j\ k(V(t_i\ u),V(t_j\ v))\cdot k(u,v) \\
&\geq 0.
\end{aligned}$$

For the last step, we proceed as follows. By the induction hypothesis, k is positive definite on both $\mathfrak{B}_\beta \cap \mathfrak{B}_m$ and $\mathfrak{B}_\gamma \cap \mathfrak{B}_m$. Hence the function

$$h : ((\mathfrak{B}_\beta \cap \mathfrak{B}_m) \times (\mathfrak{B}_\gamma \cap \mathfrak{B}_m)) \times ((\mathfrak{B}_\beta \cap \mathfrak{B}_m) \times (\mathfrak{B}_\gamma \cap \mathfrak{B}_m)) \to \mathbb{R}$$

defined by

$$h((u,y),(v,z)) = k(u,v)\cdot k(y,z)$$

is positive definite, since h is a tensor product of positive definite kernels [15]. Now consider the set

$$\{(u,V(t_i\ u)) \mid i=1,\ldots,n \text{ and } u\in supp(t_i)\}$$

of points in $(\mathfrak{B}_\beta \cap \mathfrak{B}_m) \times (\mathfrak{B}_\gamma \cap \mathfrak{B}_m)$ and the corresponding set of constants

$$\{c_{i,u} \mid i = 1, \ldots, n \text{ and } u \in \mathit{supp}(t_i)\},$$

where $c_{i,u} = c_i$, for all $i = 1, \ldots, n$ and $u \in \mathit{supp}(t_i)$.

3. Let $\alpha = \alpha_1 \times \cdots \times \alpha_m$. Suppose that $t_i = (t_i^{(1)}, \ldots, t_i^{(m)})$, for $i = 1, \ldots, n$. Then

$$\begin{aligned}
&\sum_{i,j \in \{1,\ldots,n\}} c_i \; c_j \; k(t_i, t_j) \\
&= \sum_{i,j \in \{1,\ldots,n\}} c_i \; c_j (\sum_{l=1}^{m} k(t_i^{(l)}, t_j^{(l)})) \\
&= \sum_{l=1}^{m} \sum_{i,j \in \{1,\ldots,n\}} c_i \; c_j \; k(t_i^{(l)}, t_j^{(l)}) \\
&\geq 0,
\end{aligned}$$

by the induction hypothesis. □

Experimental Comparison of Graph-Based Relational Concept Learning with Inductive Logic Programming Systems

Jesus A. Gonzalez[1], Lawrence B. Holder[2], and Diane J. Cook[2]

[1] Instituto Nacional de Astrofisica, Optica y Electronica (INAOE), Puebla, Mexico
jagonzalez@inaoep.mx
[2] Department of Computer Science and Engineering, University of Texas at Arlington
Arlington TX 76019 USA
{holder,cook}@cse.uta.edu

Abstract. We compare our graph-based relational concept learning approach "SubdueCL" with the ILP systems FOIL and Progol. In order to be fair in the comparison, we use the conceptual graphs representation. Conceptual graphs have a standard translation from graphs into logic. In this way, we introduce less bias during the translation process. We experiment with different types of domains. First, we show our experiments with an artificial domain to describe how SubdueCL performs with the conceptual graphs representation. Second, we experiment with several flat and relational domains. The results of the comparison show that the SubdueCL system is competitive with ILP systems in both flat and relational domains.

1 Introduction

In this paper we show the effectiveness of graph-based relational concept learning through a comparison with the Inductive Logic Programming (ILP) systems FOIL [1] and Progol [8]. We chose ILP systems because they have dominated the area of relational concept learning and we wanted to compare with the best. In order to be fair in the comparison we use the conceptual graphs representation introduced by John Sowa [11]. We need conceptual graphs because there is a standard translation from conceptual graphs into logic (which we use to convert our graphs into logic to test with FOIL and Progol) and in this way, we introduce less bias during the translation from graphs into logic. We will describe conceptual graphs in section 2.1. In section 2.2 we describe the Subdue system, which is the predecessor of our graph-based relational concept learning system SubdueCL. In section 3, we present SubdueCL including a description of its algorithm. In section 4 we describe the experiments and results performed with different types of domains. The first experiment involves an artificial domain, where we find out how the SubdueCL system behaves when using the conceptual graphs representation versus non-conceptual graphs. The second set of experiments focuses on a comparison of SubdueCL with the ILP systems using flat domains. Finally, in the third part of the experiments section, we perform a comparison using structural domains. The results of the comparison show that SubdueCL is

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 84–100, 2003.

competitive with ILP systems in both types of domains showing that a graph-based representation can be effectively used for the concept learning task. An advantage of a graph representation is that it is very powerful so that complex structural domains like the chemicals compounds are easy to express. Another advantage of a graph representation is that it is more natural for visual interpretation.

2 Related Work

In this section we present the work related to the area of learning using graphs. We start with a brief description of conceptual graphs in section 2.1, then we continue with a description of graph-based discovery in section 2.2.

2.1 Conceptual Graphs

Conceptual graphs are important for our empirical analysis. There is a standard translation from conceptual graphs into logic and vice versa. We want to use conceptual graphs for the comparison with ILP systems so that we do not introduce any bias while converting graphs into the logic representation used by ILP systems. In section 4 we present some experiments that indicate the utility of using conceptual graphs for the comparison with ILP systems.

Conceptual Graphs (CGs) are a logic-based knowledge representation derived from semantic networks [7] and Peirce existential graphs [11]. CGs are being used in different areas such as natural language processing, information retrieval and expert systems.

Formally, a conceptual graph [11] $G = (R, C, U, lab)$ is a bipartite, connected, and finite graph. R and C are the relation and concept vertices respectively. Relations and concepts are organized according to their type, which is taken from the sets of relation and concept types T_R and T_C respectively. A marker M is defined as a valid instantiation of a concept type from T_C (i.e., "Cat" is a marker for type "Animal"). U is the set of edges, where the edges of a particular relation from R are totally ordered. The vertex labels are defined by the mapping lab where: if $c \in C$, then $lab(c) \in T_c \times M \cup \{*\}$. If $r \in R$, then $lab(r) \in T_r$.

Figure 1 shows an example of a conceptual graph expressing "a cat is on a mat." In figure 1, x:y denotes that y is a marker of type x.

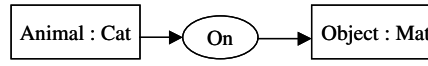


Fig. 1. Conceptual Graph Example

Concept vertices are used to represent entities, attributes, states and events. Relation vertices are used to interconnect concept vertices.

2.2 Subdue

Subdue [2,3] is a relational learning system used to find substructures (subgraphs) that appear repetitively in the graph representation of databases. Subdue starts by looking

for the substructure that best compresses the graph using the Minimum Description Length (MDL) principle [10], which states that the best description of a data set is the one that minimizes the description length of the entire data set. In relation to Subdue, the best description of the data set is the one that minimizes:

$$I(S) + I(G|S) . \tag{2.1}$$

where S is the substructure used to describe the input graph G, $I(S)$ is the length (number of bits) required to encode S, and $I(G|S)$ is the length of the encoding of graph G after being compressed using substructure S.

After finding the first substructure, Subdue compresses the graph and can iterate to repeat the same process. Subdue is able to perform an inexact match that allows the discovery of substructures whose instances have slight variations. Another important characteristic of Subdue is that it allows the use of background knowledge in the form of predefined substructures.

The model representation used by Subdue is a labeled graph. Objects are represented by vertices, while relations are represented by edges. Labels are used to describe the meaning of edges and vertices. When we work with relational databases, each row can be considered as an event. Events may also be linked to other events through edges. The event attributes are described by a set of vertices and edges, where the edges identify the specific attributes and the vertices specify the value of that attribute for the event.

Subdue uses a computationally-constrained beam search to find substructures. A substructure is a subgraph contained in the input graph. The algorithm starts with a single vertex as the initial substructure and at each iteration expands the instances of that substructure by adding an edge in every possible way, generating new substructures that might be considered for expansion. Section 3 provides more detail into the discovery algorithm and its use in graph-based concept learning.

3 Graph-Based Concept Learning

The main challenge in adding concept-learning capabilities to Subdue is the inclusion of "negative" examples into the process. Substructures that describe the positive examples, but not negative examples, are likely to represent the target concept. Therefore, the Subdue concept learner (which we will refer to as SubdueCL) accepts positive and negative examples in graph format.

Since SubdueCL is an extension to Subdue, it uses Subdue's core functions to perform graph operations, but the learning process is different. SubdueCL works as a supervised learner by differentiating positive and negative examples using a set-covering approach instead of graph compression. The hypothesis found by SubdueCL consists of a disjunction of substructures. SubdueCL forms one of these substructure disjuncts in each iteration. Positive example graphs that are described by the substructure found in a previous iteration are removed for subsequent iterations.

3.1 Substructure Evaluation

The way in which SubdueCL decides if a substructure will be part of the concept is different from Subdue. SubdueCL uses an evaluation formula to give a value to all the

generated substructures. This formula assigns a value to a substructure according to how well it describes the positive examples (or a subset of the positive examples) without describing the negative examples. A substructure covers an example if the substructure matches a subgraph of the example. This graph match can be inexact, as controlled by a user-defined match threshold, and is constrained to run in time polynomial in the size of the graphs. Positive examples covered by the substructure increase the substructure value, while negative examples decrease its value. Positive examples that are not covered and the negative examples covered by the substructure are considered errors, because the ideal substructure would be one covering all the positive examples without covering any negative example. The substructure value is calculated as shown in formula 3.1:

$$value = 1 - Error \ . \tag{3.1}$$

where the error is calculated with respect to the positive and negative examples covered by the substructure using the following formula:

$$Error = \frac{\#PosEgsNotCovered + \#NegEgsCovered}{\#PosEgs + \#NegEgs} \ . \tag{3.2}$$

#PosEgsNotCovered is the number of positive examples not covered by the substructure, and #NegEgsCovered is the number of negative examples covered by the substructure. #PosEgs is the number of positive examples remaining in the training set (the positive examples that have already been covered in a previous iteration were removed from the training set), and #NegEgs is the total number of negative examples, which does not change, because negative examples are not removed from the training set.

The only problem found with the use of formula 3.2 is that when two substructures have the same error, we would like to prefer the substructure covering more positive examples. For instance, suppose we have 10 positive examples and 10 negative examples. Substructure *S1* covers 5 positive examples and 0 negative examples, and substructure *S2* covers 10 positive examples and 5 negative examples. In this case both substructures have an error of 1/4 according to formula 3.2, but we prefer SubdueCL to select *S1*, because it does not cover any negative examples. In order to do this we need to assign a negative weight (say *k*) to the negative errors. After some algebraic manipulation we express the adjusted error with formula 3.3, where *k* is the negative weight. The default value for k is 3, but any value greater than or equal to 2 will effect the desired preference.

$$\frac{AdjustedError = \#PosEgsNotCovered + (k * \#NegEgsCovered - \#NegEgs * (1-k))}{\#PosEgs + \#NegEgs} \ . \tag{3.3}$$

Now the adjusted error of *S1* and *S2* according to formula 3.3 (and with the default value of k = 3) is 5/4 and 7/4 respectively. Using this formula, SubdueCL will prefer *S1* over *S2*.

3.2 SubdueCL Algorithm

The SubdueCL algorithm is shown in figures 2 and 3. The main function takes as parameters the positive examples G_p, the negative examples G_n, the *Beam* length

(since SubdueCL's search algorithm is a beam search), and a *Limit* on the number of substructures to include in its search. The main function makes calls to the SubdueCL function in order to form the hypothesis *H* that describes the positive examples.

```
Main(Gp, Gn, Limit, Beam)
  H = {}
  repeat
    repeat
      BestSub = SubdueCL(Gp, Gn, Limit, Beam)
      if BestSub = {}
        then Beam = Beam * 1.1
    until(BestSub ≠ {})
    Gp = Gp - {p ∈ Gp | BestSub covers p}
    H = H + BestSub
  until Gp = {}
  return H
end.
```

Fig. 2. SubdueCL's Main Algorithm

A substructure is added to *H* after each call to the SubdueCL function (as a step of the set covering approach). In the case that SubdueCL returns NULL (and there are still positive examples to cover), the *Beam* is increased by a 10% so that a larger search space can be explored during SubdueCL's search. We chose to increase the beam by 10%, because that value was enough to find a substructure in the next iteration in most experiments. Also, after SubdueCL finds a substructure, the positive examples covered by it are removed from the positive graph.

Figure 3 shows the SubdueCL function, which receives as input the positive and negative examples (G_p and G_n) and builds a *ParentList* containing a substructure for each vertex in the positive graph (G_p) with a different label, but keeping only as many substructures as the length of the *Beam*. The "*mod* Beam" qualifier means that the lists keep only as many substructures as the *Beam* size. Each of those substructures in the *ParentList* is then expanded by one edge (adding an edge to two existing vertices in the substructure) or one vertex and an edge (adding the vertex and an edge from the existing vertex to the new one) in all possible ways by the *Expand* function and then evaluated using the *Evaluate* function according to equation 3.2 presented earlier. Those substructures that cover at least one positive example are selected by the *CoversOnePos* function and kept in the *BestList*, which is limited to the *Beam* size (the substructures are kept ordered, and the best substructure can be found at the head of the list). A *ChildList* keeps all the substructures that were obtained from the expansion of the substructures in the *ParentList* and is also limited by the *Beam* size.

The *Limit* parameter is used to constrain the number of expanded substructures, but if the *BestList* is empty after expanding *Limit* substructures from the *ParentList*, the limit is increased by 20% until one is found. We chose to increase this limit by 20%, because it was usually enough to find a positive substructure in the next trial when working with our experiments. Finally, the SubdueCL function returns the best of *BestList*. It is important to mention that all the lists are ordered according to the substructures' values.

```
SubdueCL(Gp, Gn, Limit, Beam)
  ParentList=(substructures of one vertex in Gp) mod Beam
  repeat
    BestList = {}
    Exhausted = TRUE
    i = Limit
    while ((i>0) and (ParentList ≠ {}))
      ChildList = {}
      foreach substructure S in ParentList
        foreach expanded substructure C in Expand(S)
          Evaluate(C, Gp, Gn)
          if CoversOnePos(C,Gp)
            then BestList = BestList ∪ {C}
          ChildList = (ChildList ∪ C) mod Beam
          i = i - 1
        endfor
      endfor
      ParentList = ChildList
    endwhile
    if BestList = {} and ParentList ≠ {}
      then Exhausted = FALSE
           Limit = Limit * 1.2
  until(Exhausted = TRUE)
  return first(BestList)
end.
```

Fig. 3. SubdueCL Algorithm

The version of SubdueCL just presented may return hypotheses inconsistent with the training examples. The only variation to produce a consistent version of the SubdueCL algorithm is to only consider substructures covering no negative examples in the *BestList*.

3.3 Relational Aspects of the Approach

The approach followed with the SubdueCL method is very suitable for relational domains. The approach is based in a graph-based knowledge representation, which allows modeling complex relational domains like the carcinogenesis domain presented in section 4.2.2 in a natural and easy to understand way for every type of user. Another advantage of our relational approach (also based in the knowledge representation), is that the search engine of SubdueCL naturally follows the relations between vertices and edges of the input graph (SubdueCL's substructures must be connected graphs). In this way, SubdueCL uses the graph edges (which represent relations) that link objects or concepts (represented by vertices) in order to expand the substructures that might be included in the hypothesis that is being searched for as the target concept. This forces the substructures to incorporate connecting contextual information

that, while not necessary to improve accuracy on the training data, may improve the quality and predictive accuracy of the resulting hypothesis.

4 Experiments

In this section we present experimental results to evaluate SubdueCL's performance with different types of graphs. First, we use an artificial domain to see if SubdueCL learns known concepts. Then, we compare SubdueCL with ILP systems. In this comparison with ILP systems, we test SubdueCL with flat and relational domains.

4.1 Artificial Domain

The purpose of this experiment is to evaluate the effectiveness of SubdueCL in learning known relational concepts and the use of the conceptual graph representation. An artificial domain is ideal for this, because we can control the experiment's environment. The basic idea of this experiment is that we embed a concept in a set of graphs and run SubdueCL to find that concept. In this experiment we used the inconsistent version of SubdueCL. We consider two types of graphs: normal graphs and conceptual graphs. *Normal graphs* consist of labeled vertices linked with labeled edges, where vertices represent concepts and edges represent relations among them. *Conceptual graphs*, as presented in section 2.1, are composed of labeled vertices of two types, concept vertices and relation vertices, that are linked with edges (unlabeled for the case of only binary relations).

As we mentioned in section 2.1, there is a standard translation of conceptual graphs into logic and vice versa. We can use this property when we compare SubdueCL with ILP systems; in this way we use the same procedure to translate our graphs into logic or the ILP rules to graphs while minimizing bias during the knowledge representation process. This is the motivation for showing that SubdueCL produces similar results using normal and conceptual graphs.

In order to make this evaluation with SubdueCL, some experiments were conducted using an artificial domain. The artificial domain consists of a set of graphs. Some of those graphs represent positive examples and others negative examples. Positive example graphs are distinguished from the negative examples, because a known substructure is embedded into them, but not into the negative examples.

We performed a total of 13 tests varying the similarity between the positive and negative substructures and the graph's sizes. As an example of the graphs used for this experiment, figure 4 shows the positive and negative substructures that were embedded in the positive and negative examples of tests 5 and 6. Figure 5 shows the substructure that SubdueCL learned from those examples using normal and conceptual graphs.

From the results of this experiment, we could see that SubdueCL finds the positive embedded substructure (or a subgraph of it) and produces similar results using both normal and conceptual graphs (as we can see in figure 5 for test number 6). The conceptual graph shown in the result for test number 6 corresponds to a subgraph of the result for normal graphs. For test number 6, the only difference between the graphs is that the conceptual graph result does not contain the vertex "v4". For the result of test number 5, the conceptual graph is not a subgraph of the normal graph but they are very similar.

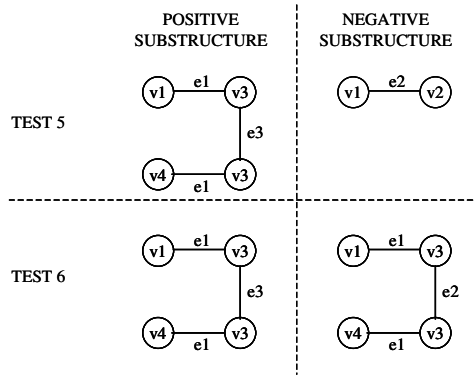


Fig. 4. Positive and Negative Substructures Embedded in the Graphs for Tests 5 and 6

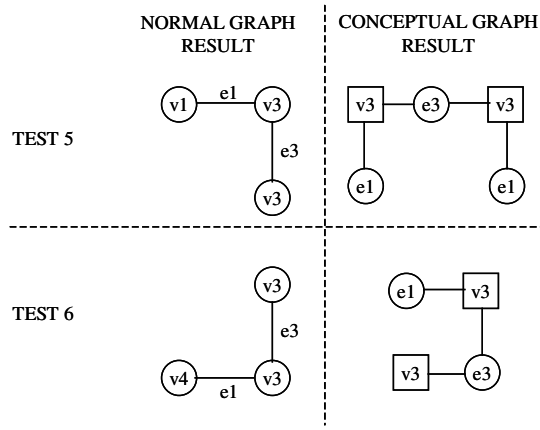


Fig. 5. Result for Tests 5 and 6 with Normal and Conceptual Graphs

From the similarity of the results using conceptual and normal graphs for this experiment, we can conclude that SubdueCL discovers almost the same substructures. The difference between the results with normal and conceptual graphs in most of the cases is that the conceptual graph result is a subgraph of the normal graph result. This means that we can use a conceptual graphs representation with SubdueCL for the comparison with ILP systems.

4.1.1 10-Fold Cross Validation

The purpose of this experiment is to evaluate SubdueCL's accuracy in classifying unseen examples from a testing set after learning from a set of training examples using normal and conceptual graphs. We performed a 10-fold cross validation with SubdueCL, also varying the graph size, with the positive and negative substructures of tests 5 and 6 shown in figure 4. We chose tests 5 and 6 to differentiate SubdueCL's performance when the positive and negative substructures are very different (test 5) and when the positive and negative substructures are very similar (test 6). The experiments consisted of using a different 90% of the examples for training and the rest

for testing, and evaluating SubdueCL according to its performance on the testing examples.

Table 1 shows the details of the experiment. The field "Test" refers to the positive and negative substructures used (from figure 4). The fields "# Examples Pos/Neg" and "# Vertices/Edges" refer to the number of positive and negative examples used in the test, and the number of vertices and edges in each example. The field "# Errors Pos/Neg" refers to the number of positive and negative errors made by SubdueCL during the testing phase for the experiment (cumulative error from the cross validation). The field "Accuracy" presents the percent accuracy of SubdueCL averaged over each fold's testing.

For the experiments with normal graphs we used the default value of the beam size, which is 4. For the experiments with conceptual graphs we had to increase the beam size up to 30 for large graphs (we repeatedly increased the beam until the accuracy achieved could not be improved). We had to do this because the search space that SubdueCL needs to explore with conceptual graphs is larger than that of normal graphs (conceptual graphs contain more vertices and edges).

From the results in table 1, we can see SubdueCL is less accurate using conceptual graphs. A reason for this is that some of the substructures learned by SubdueCL with conceptual graphs are more general (see test 6, from figure 5) and then, the concept learned from the examples might be incomplete. This will cause more errors at testing time. Another reason for this is that with conceptual graphs the substructures learned tend to have fewer concept vertices than with normal graphs. These results show us that SubdueCL will perform with a higher accuracy for domains in which the positive examples are very different from the negative examples. From the results we can also see that SubdueCL performs with higher accuracy using normal graphs than conceptual graphs. This accuracy is worse in the presence of noise. Although SubdueCL produces lower accuracy using conceptual graphs, we will see that SubdueCL is still competitive with ILP systems using the conceptual graphs representation as mentioned in section 4.1.

Table 1. Results of the 10-Fold Cross Validation of SubdueCL in the Artificial Domain

Type of Graphs	Test	#Examples Pos/Neg	# Vertices/Edges	# Errors Pos/Neg	Accuracy
Normal	5	50/50	20/40	10/0	90%
	5	50/50	50/100	5/3	92%
	6	50/50	20/40	2/8	90%
	6	50/50	50/100	6/0	94%
Conceptual	5	50/50	60/80	9/4	87%
	5	50/50	150/200	9/3	88%
	6	50/50	60/80	12/2	86%
	6	50/50	150/200	17/1	82%

From other experiments conducted in the artificial domain we found that SubdueCL produces more substructures to represent a concept as the graph density (number of edges divided by the number of vertices) increases (which also introduces noise). Finally we studied the learning curve produced by SubdueCL and found that it is able to learn a domain with a low number of training examples if the positive and negative examples are very different [4].

4.2 Comparison of SubdueCL with ILP Systems

Logic-based systems have dominated the area of relational concept learning, especially Inductive Logic Programming (ILP) systems [9]. However, first-order logic can also be represented as a graph, and in fact, first-order logic is a subset of what can be represented using graphs [11]. Therefore, learning systems using graphical representations have the potential to learn richer concepts if they can handle the increased size of the hypothesis space. In this section we compare SubdueCL to two ILP systems: FOIL [1] and Progol [8]. Quantitative comparisons indicate that the graph-based learner is competitive with the logic-based learners. We conducted experiments in structural (relational) and non-structural domains in order to capture the advantages and disadvantages of SubdueCL in these types of domains. For these experiments we used the conceptual graph representation (as presented in section 2.1).

4.2.1 Non-relational Domains

Five non-relational domains were used to compare FOIL, Progol and SubdueCL: golf, vote, diabetes, credit and tic-tac-toe. The golf domain is a trivial domain used to demonstrate machine learning concepts, typically decision-tree induction [1]. The domain consists of fourteen examples (10 positive or play golf, 4 negative) and four attributes, two discrete and two continuous, with no missing values. The vote domain is the Congressional Voting Records Database available from the UCI machine learning repository [6]. The diabetes domain is the Pima Indians Diabetes Database, and the credit domain is the German Credit Dataset from the Statlog Project Databases, also available from the UCI repository. For the Tic-Tac-Toe domain, 958 examples were exhaustively generated. Positive examples are those board configurations where "X" wins the game, and negative examples are those board configurations where "X" loses or the game is tied.

Table 2. Percent Accuracy Results on Non-Relational Domains: Golf, Vote, Diabetes, Credit, and Tic-Tac-Toe

	Golf	Vote	Diabetes	Credit	T-T-T
FOIL	66.67	93.02	70.66	68.60	100.0
Progol	66.67	94.19	63.68	63.20	100.0
SubdueCL	66.67	94.65	65.79	63.50	100.0

The accuracy values shown in Table 2 are the results of averaging the individual accuracies found using a 10-fold cross validation. The only domain where a 3-fold cross validation was used is the golf domain, because it consisted of only fourteen examples. In the case of the diabetes domain FOIL produced the best result with an accuracy of 70.66%, then SubdueCL with an accuracy of 65.79% and finally Progol with 63.68% accuracy. The low accuracy values are due to the continuous values of the attributes of the domain. SubdueCL does not yet learn ranges of values. From these results we can see that FOIL can better deal with continuous values. For the other domains, where we have either only discrete attributes or a combination of discrete and continuous values (Golf, Vote, Credit, and Tic-Tac-Toe domains), SubdueCL is competitive (better in some cases) in terms of accuracy with FOIL and Progol. A disadvantage of SubdueCL is that it does not have an effective way to deal

with continuous values; that is, it cannot represent or output a substructure that refers to a range of values. This effect is most pronounced in the Diabetes domain, where most of the fields are continuous, and in the credit domain that consists of a combination of discrete and continuous values.

These results show that for non-relational domains SubdueCL is competitive and even better than ILP systems if the domain does not contain continuous values. The differences between the accuracies of the systems in the diabetes domain are significant. In the case of the credit domain, the differences between the accuracies of SubdueCL versus FOIL and FOIL versus Progol are significant. In order to be more competitive with ILP systems in this type of domain, we need to add the capability to deal with ranges of numbers.

4.2.2 Relational Domains

The three relational domains used in this study are illegal chess endgames, carcinogenesis and website hyperlink structure. Representations and results for the three domains are discussed in the following sections.

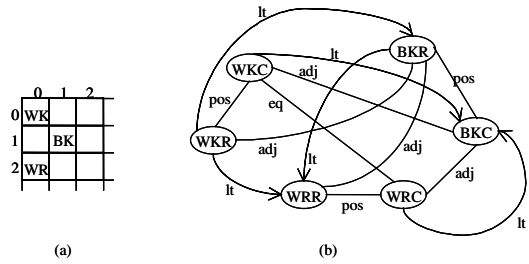


Fig. 6. An Example from the Chess Domain. (a) Board configuration and (b) SubdueCL's graphical representation of the example

Chess Domain.

The chess domain consists of 20,000 examples of row-column positions for a white king, white rook and black king such that the black king is in check (positive) or not (negative). Therefore, if white's turn is next, then the positive examples are illegal configurations. The relational information in this domain consists of adjacency relations between chess-board positions and less-than or equal relations between row and column numbers (0-7). Both FOIL and Progol extensionally define the three relations.

Figure 6b shows the graph representation used for the chess domain example in figure 6a. Each piece is represented by two vertices corresponding to the row and column of the piece, connected by a position relation (e.g., WKC stands for white king column).

Due to computational constraints only a subset (5,000 examples) of the entire database was used for the 10-fold cross validation. The accuracy results are 99.74% for Progol, 99.34% for FOIL, and 99.74% for SubdueCL. The difference in error between SubdueCL and FOIL (which is the same as that between Progol and FOIL) is of 0.4% (+/- 0.7242%) with a confidence of 94.27%, and we are 89.27% certain (by ANOVA results) that the confidence value of the difference is correct. In terms of number of rules, Progol learned 6 rules, FOIL learned 11 rules, and SubdueCL learned 7 rules (substructures). The three systems perform comparably in this domain.

Carcinogenesis Domain.
The PTC (Predictive Toxicology Challenge) carcinogenesis databases contain information about chemical compounds and the results of laboratory tests made on rodents in order to determine if the chemical induces cancer to them. The data in this database comes from the predictive toxicology evaluation project conducted by the National Institute of Environmental Health Sciences (NIEHS). We applied SubdueCL to the first two databases PTC1 and PTC2 but the most recent result comes from "The Predictive Toxicology Challenge 2000 – 2001" or PTC3 (see http://www.informatik.uni-freiburg.de/~ml/ptc/ for more information about the challenge).

Each example is composed of all the information related to the compound: the atoms that it contains, the element, type and charge of each atom, the bonds between atoms, the groups that are formed by the atoms and counters of groups of the same type (i.e., number of amine groups).

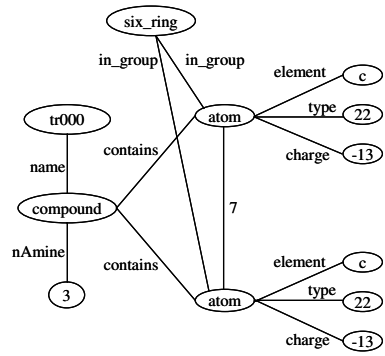


Fig. 7. Graph representation of a chemical compound from the Cancer Domain

The graph representation used for the cancer domain is shown in figure 7 (actual examples are much larger than this example). In the figure we can see that compounds are linked to their atoms with edges labeled "contains." Each atom vertex is connected to three vertices that describe the atom's element, type, and charge, linked with the edges "element," "type," and "charge" respectively. Two atoms can be connected with a bond edge. Bond edges are labeled with the type of bond, which is an integer number (1 = Single, 2 = Double, 3 = Triple, 4 = Aromatic, 5 = Single or Double, 6 = Single or Aromatic, 7 = Double or Aromatic, and 8 = Any). Groups are represented with vertices and connected to the vertices of all the atoms participating in the group with edges labeled as "in_group." We also included counters of the number of groups of the same type contained by the compound. Counters were represented with a vertex labeled with the number of groups (of the same type, e.g., amine) and linked with an edge with the name of the group preceded by an "n" (e.g., nAmine) to the compound. We did not include the short-term assays or predictive tests as used in the previous experiments, because they were not available for the new dataset.

We created a program to read all the information related to the compounds from an input file and generate an output graph file. The output graph file consists of positive and negative examples identified by their category (male rats MR, female rats FR, male mice MM, or female mice FM).

We created four training files and four testing files (i.e., a training file for male rats with its corresponding testing file). The results of the challenge were released in August 2001, and SubdueCL achieved maximum performance for the male rats category among other challenge submissions for the same category. This is an encouraging result that shows how SubdueCL is successful in this type of structural domain. Figure 8 shows one of the substructures found by SubdueCL in this domain [5]. As we can see from the results, SubdueCL was able to find substructures of chemical compounds that are correlated with their tendency to cause cancer. These substructures are now under evaluation by toxicological experts to determine their relevancy.

We also performed a 10-fold cross validation with the training data with the results shown in table 3. As we can see, the three systems obtained very similar accuracies for this domain. The differences between the accuracies of the systems in this domain are not significant. The accuracy results are low due to the difficulty of the task.

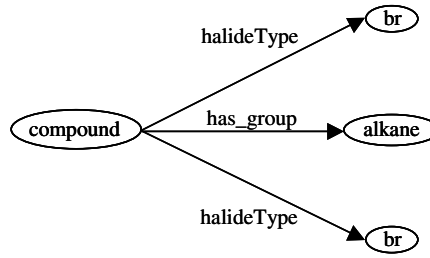


Fig. 8. A compound that contains two "halide" groups of type "br" and that also has an "alkane" group may cause cancer

Table 3. Accuracy Results of a 10-Fold Cross Validation in the Carcinogenesis Domain

	Male Rats	Female Rats	Male Mice	Female Mice
SubdueCL	64.00%	72.14%	68.80%	65.47%
Progol	63.81%	70.48%	65.48%	64.29%
FOIL	62.38%	70.95%	68.80%	65.95%

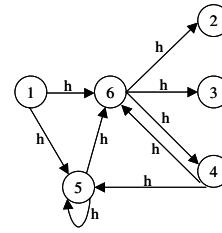


Fig. 9. A Graph form the Web Domain created with the Hyperlink structure option

Web Domain.
The Web domain consists of graphs created from web sites. At the moment we have three options for the information that these graphs contain:

- Hyperlink structure
- Hyperlink structure + Page's title
- Hyperlink structure + Page's content

Figure 9 shows an example of a graph with the hyperlink structure option. Figure 10 shows an example of a graph created with the hyperlink structure plus the page's content.

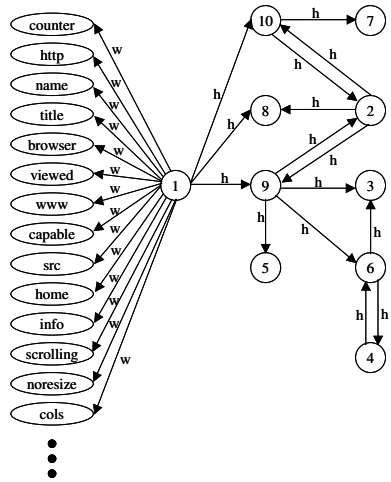


Fig. 10. A Graph form the Web Domain created with the Hyperlink Structure + Page's Content Option

We are using a Perl program to extract this information and convert it into its graph representation. The program extracts the pages' hyperlink structure and converts it to a graph. In the case of the structure option, vertices have the number of the page and edges are labeled as "hyperlink" or "h" as in figure 9. If the page's content or title is considered, each word is represented as a vertex and is linked to the page number to which it belongs with an edge labeled as "word" or "w" as in figure 10.

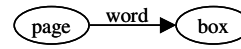


Fig. 11. A Substructure Found in the Web Domain (Professors-Students)

The first experiment that we made for the Web domain consisted in differentiating the Web pages of professors and students. We considered the web sites of the professors and students of the Computer Science Department of the University of Texas at Arlington. We created graphs for all the professors and students and selected those graphs having at least five vertices. We made the professors' web pages our positive examples and the students' web pages our negative examples. For this initial experi-

ment we chose the web page structure + page's content option, because we wanted to give as much information as possible to SubdueCL and observe its behavior. SubdueCL was able to find a very small substructure that could differentiate the positive examples from the negative examples. This substructure is shown in figure 11 and says that every professor has in their web page the word "box", but students do not have this word in their web pages. This is true because the word "box" is part of the address field of the professors' web pages.

Table 4. Accuracy Results for the Web Domains (Professors-Students and ComputerStores-Professors)

	Professors-Students	**ComputerStores-Professors**
SubdueCL	84.50%	75.93%
Progol	63.64%	78.43%
FOIL	61.59%	61.22%

We also performed a 3-fold cross validation for this experiment. We used 24 positive examples (professors web pages) and 23 negative examples (students web pages). Results of this test for the three systems can be found in table 4. As we can see, SubdueCL achieved 84.5% accuracy, (and performed better than Progol and FOIL). This means that the predictive accuracy of a hypothesis produced by SubdueCL in the web domain is not very high (only 84.5%) when using the hyperlink structure + page's content. We performed the same 3-fold cross validation experiment with Progol and FOIL obtaining an accuracy of 63.64% and 61.59 respectively, which is even lower than SubdueCL's accuracy. The differences between the accuracies of the systems in this domain (Professors-Students) are significant.

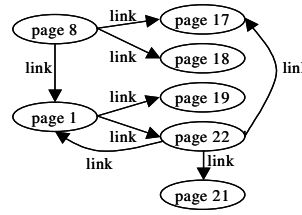


Fig. 12. A Substructure Found in the Web Domain (Computer Stores - Professors)

For the second experiment in the web domain we used the hyperlink structure only. We chose this option, because we wanted to learn only structure without considering the web pages' content so that SubdueCL did not learn a substructure with a word (or set of words) describing the domain as in the previous experiment. We chose the structure graphs of web sites of computer stores as our positive examples and the structure graphs of web sites of professors as our negative examples. We searched the web for computer stores and created graphs for their web sites. As in the previous experiment, we chose only those graphs having at least five vertices. Figure 12 shows a substructure found in this domain. This substructure covered 24 of a total of 29 positive examples without covering any of the negative examples. Although not all the examples covered by this substructure had the same interpretation, for most of them the following explanation applies: Node 8 represents a category of products

linked to three subcategories of products represented by nodes 1, 17, and 18. Nodes 19 and 22 represent either more specific subcategories derived from the subcategory represented by node 1 or specific products. Node 21 represents a specialization of the subcategory represented by node 22.

We also performed a 3-fold cross validation for this test. We used 31 positive examples (computer stores' pages) and 25 negative examples (professors' pages). Results of this test can be seen in table 4. SubdueCL achieved a predictive accuracy of 75.93% using the hyperlink structure method. This means that the predictive accuracy of SubdueCL using hyperlink structure is even lower than the predictive accuracy achieved using the hyperlink structure + pages' content option. We performed the same experiment with Progol and FOIL, and achieved an accuracy of 78.43% and 61.22% respectively. Progol obtained a higher accuracy than SubdueCL. This means that Progol performs slightly better for the experiments containing only hyperlink structure. FOIL obtained the lowest accuracy for this test. The differences between the accuracies of the systems in this domain (ComputerStores-Professors) are not significant.

The results in the web domain tell us that SubdueCL is able to learn how to differentiate between the graph structure of different classes of web sites and is still competitive with Progol on some aspects of this task. Note that in the case that one class subsumes the other (as is the case with graphs of the structure of professors' web sites as the positive examples and graphs of the structure of computer stores web sites as the negative examples), we need to make the super class to be the positive examples and the subclass to be the negative examples.

5 Conclusion

In this research we compared our graph-based relational concept learning approach with the ILP systems FOIL and Progol through a set of experiments. The experiments made with an artificial domain show that SubdueCL is able to successfully learn a concept from examples using conceptual graphs, which is important because graphs have a standard translation into logic that we use to produce the logic representation needed by the ILP systems. In this way we introduce less bias during the comparison. We compared SubdueCL to the ILP systems FOIL and Progol in relational and non-relational domains. The results of this comparison for non-relational domains show that SubdueCL either outperforms or obtains the same performance as FOIL and Progol when the domain does not contain only continuous values. For the case of relational domains SubdueCL outperformed FOIL and obtained the same accuracy as Progol. This shows that SubdueCL is competitive with these ILP systems, but has the advantage of having a more natural way to represent these relational domains. These results show that SubdueCL successfully learns in relational domains. In our future work we need to test SubdueCL with more relational databases like in the 2001 Knowledge Discovery in Databases (KDD) Cup that focuses on data from genomics and drug design.

There are some enhancements that we need to make to SubdueCL to make it more competitive with ILP systems. First, we need to give SubdueCL the ability to express ranges of values. This would be very useful for domains that involve continuous variables. In this type of domain SubdueCL could find substructures containing vertices

whose value expresses a range of values that the continuous variable can take. Second, we need to allow SubdueCL to express that the label of one vertex is the same as another vertex, and also that the numeric label of one vertex is less than or greater than the numeric label of another vertex. Third, we need to find a representation able to describe recursive structures. This will be part of our future work.

Acknowledgements

This work was supported by the National Science Foundation grant 0097517.

References

1. Cameron-Jones, R. M., and J. R. Quinlan. (1994). Efficient Top-down Induction of Logic Programs. *SIGART Bulletin*, 5, 1:33-42.
2. Cook, D. J., and L. B. Holder. (1994). Substructure Discovery Using Minimum Description Length and Background Knowledge. *Journal of Artificial Intelligence Research*. 1:231-55.
3. Cook, D. J., and L. B. Holder. (2000). Graph-Based Data Mining. *IEEE Intelligent Systems*, 15(2):32-41.
4. Gonzalez, Jesus A. (2001). *Empirical and Theoretical Análisis of Relational Concept Learning Using a Graph-Based Representation*. Doctoral dissertation, Department of Computer Science, University of Texas at Arlington.
5. Gonzalez, Jesus A., L. B. Holder, and Diane J. Cook. (2001). Application of Graph-Based Concept Learning to the Predictive Toxicology Domain. *Proceedings of the Predictive Toxicology Challenge Workshop*.
6. Keogh, E., C. Blake, and C. J. Merz. (1998). *In UCI Repository of Machine Learning Databases*.
7. Lehmann, F. (1992). *Semantics Networks in Artificial Intelligence*. Oxford: Pergamon Press.
8. Muggleton S. (1995). Inverse Entailment and Progol. *New Generation Computing*, 13:245-86.
9. Muggleton, S. and C. Feng. (1992). Efficient Induction of Logic Programs. *Inductive Logic Programming*, Academic Press 281-97.
10. Rissanen, J. (1989). *Stochastic Complexity in Statistical Inquiry*. World Scientific Publishing Company.
11. Sowa, J. F. (1984). *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley.

Autocorrelation and Linkage Cause Bias in Evaluation of Relational Learners

David Jensen and Jennifer Neville

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

Abstract. Two common characteristics of relational data sets — concentrated linkage and relational auto-correlation — can cause traditional methods of evaluation to greatly overestimate the accuracy of induced models on test sets. We identify these characteristics, define quantitative measures of their severity, and explain how they produce this bias. We show how linkage and autocorrelation affect estimates of model accuracy by applying FOIL to synthetic data and to data drawn from the Internet Movie Database. We show how a modified sampling procedure can eliminate the bias.

1 Introduction

Accurate evaluation of learning algorithms is central to successful research in relational learning. The most common method for evaluating a learning algorithm is to partition a given data sample into training and test sets, construct a model using the training set, and evaluate the accuracy of that model on the test set. Separate training and test sets are used because of a widely observed bias when the accuracy of models is assessed on the original training set (Jensen & Cohen 2000).

In this paper, we show how dependence among the values of a class label in relational data can cause strong biases in the estimated accuracy of learned models when accuracy is estimated in this conventional way. In this section, we give a simple example of how estimated accuracy can be biased. In later sections, we define quantitative measures of concentrated linkage (L) and relational autocorrelation (C'), two common characteristics of relational data sets. We show how high values of L and C' cause statistical dependence between training and test sets, and we show how this dependence leads to bias in test set accuracy. In general, current techniques for evaluating relational learning algorithms do not account for this bias, although we discuss some special classes of relational data sets that are immune to this effect. We present a new family of sampling algorithms that can be applied to any relational data set, and show how it eliminates the bias.

These results indicate that accurate evaluation of relational learning algorithms will often require specialized evaluation procedures. The results also indicate that additional attention should be paid to identifying and using relational autocorrelation to improve the predictive accuracy of relational models. This paper is part of a larger

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 101–116, 2003.

study of the effects of linkage and autocorrelation on relational learning. A related paper (Jensen and Neville 2002) shows how linkage and autocorrelation affect feature selection in relational learning.

1.1 Statistical Analysis of Relational Data

Recent research in relational learning has focused on learning statistical models, including work on stochastic logic programming (Muggleton 2000), probabilistic relational models (Getoor et al. 1999), and relational Bayesian classifiers (Flach and Lachiche 1999). However, with the greater expressive power of relational representations come new statistical challenges. Much of the work on relational learning diverges sharply from traditional learning algorithms that assume data instances are statistically independent. The assumption of independence is among the most enduring and deeply buried assumptions of machine learning methods, but this assumption is contradicted by many relational data sets.

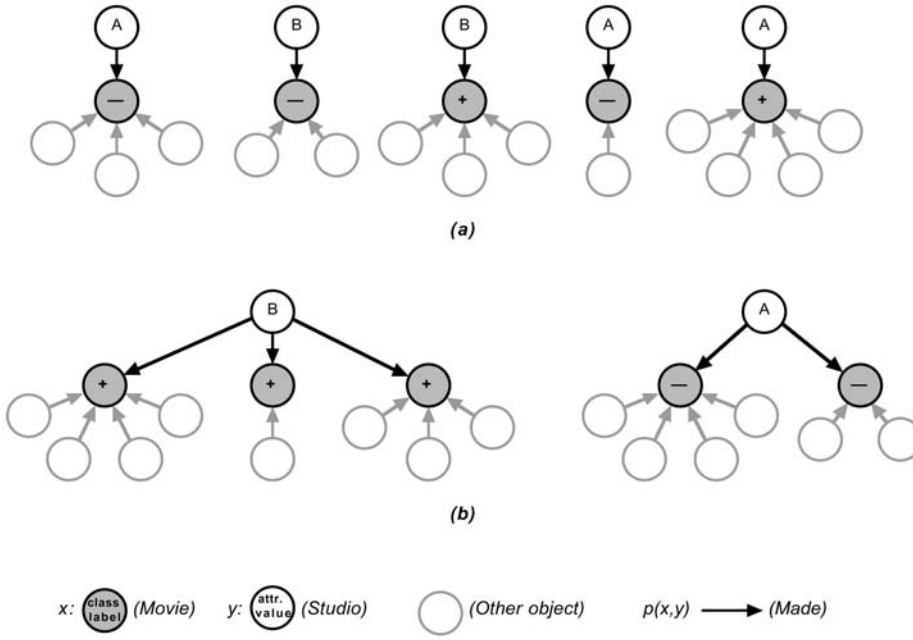


Fig. 1. Example relational data sets (a) five independent instances and (b) five dependent instances.

For example, consider the two simple relational data sets shown in Figure 1. In each set, instances for learning consist of subgraphs containing a unique object *x*, an object *y*, and one or more other objects. Objects *x* contain a class label and objects *y* contain an attribute that will be used to predict the class label of *x*. Figure 1a shows a data set where objects *x* and *y* have a one-to-one relationship and where the class labels on instances are independent. Figure 1b shows instances where objects *x* and *y* have a many-to-one relationship and where the class labels are dependent.

We spend the majority of this paper considering data sets similar in structure to Figure 1b, where each subgraph consists of multiple relations and each relation may produce statistical dependencies among the instances. For simplicity, all relations in Figure 1 are binary, but the statistical effects we investigate affect a wider range of tasks and data representations.

1.2 Simple Random Partitioning

Perhaps the most widely used evaluation technique in machine learning and data mining is the partioning of a data sample into training and test sets. Most sampling in machine learning and data mining assumes that instances are independent. In contrast, this paper examines situations where instances are not independent. Methods for sampling relational data are not well understood. In the relatively few cases where researchers have considered special methods for sampling relational data for machine learning and data mining, they have often relied on special characteristics of the data. For example, some researchers have exploited the presence of naturally occurring, disconnected subsets in the population, such as multiple websites without connections among the sites (e.g., Craven et al. 1998). We wish to evaluate classification models that operate over completely connected graphs. There is also a small body of literature on sampling in relational databases (e.g., Lipton et al. 1993), but this work is intended to aid query optimization while our interest is to facilitate evaluation of predictive models.

The most common sampling technique — *simple random partitioning* — has been taken from propositional settings and adapted for use in relational sets such as those in Figure 1. We define simple random partitioning with respect to two sets of objects X and Y and a set of paths P such that the relation $p(x,y)$ holds. We presume that objects X contain class labels and that objects X and Y both contain attributes relevant to classifying objects X. Paths are composed of k links and $k-1$ intervening objects, where $k \geq 1$. Each path represents a series of relations linking an object in X to an object in Y. For example consider the path linking two movies, m_1 and m_2, made by the same studio. The path is formed from two *Made* links, $Made(m_1,s_1)$ and $Made(m_2,s_1)$. For convenience we treat all links as undirected in order to refer to meaningful sequences of relationships as paths. We assume that paths in P are unique with respect to a given (x,y) pair; if two or more paths between x and y exist in the data, they are collapsed to a single element of P.

> **Definition:** *Simple random partitioning* divides a sample S of relational data into two subsamples S_A and S_B. The subsamples are constructed by drawing objects X from S without replacement and without reference to paths P and objects Y in S. When an individual object $x \in X$ is placed into a subsample, some set of objects $\{y_1, y_2, ...y_n\}$, such that $p(x,y_i)$, are also placed into the subsample if those objects are not already present. The object sets X_A and X_B are mutually exclusive and collectively exhaustive of objects X in S, but other objects Y may appear in both S_A and S_B.

Such a techique for creating training and test sets seems a logical extension of techniques for propositional data. However, it leaves open the possibility that a subset of objects Y may fall into both the training and test set, creating some type of dependence between the training and test sets.

1.3 An Example of Test Set Bias

Given that instances in relational data may share some objects, and thus not be independent, we should examine how such relational structure could affect accuracy estimates made using training and test sets. Below we show how relational structure and dependence among values of the class label can bias estimates of the accuracy of induced models.

We created data sets about movies and analyzed them using FOIL (Quinlan 1990). Specifically, we created and analyzed simple relational data sets whose relational structure was drawn from the Internet Movie Database (www.imdb.com). We gathered a sample of all movies in the database released in the United States between 1996 and 2001 for which we could obtain information on box office receipts. In addition to 1382 movies, the data set also contains objects representing actors, directors, producers, and studios[1]. In all, the data set contains more than 40,000 objects and almost 70,000 links. The data schema is shown in Figure 2.

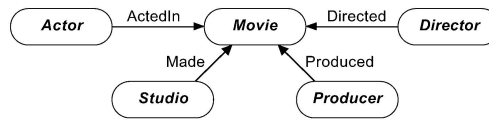


Fig. 2. A general data schema for the movie data sets.

For most of the experiments reported in this paper, we limited analysis to just two classes of objects — movies and studios. This allowed us to greatly reduce the overall size of training and test sets, thus making them feasible to analyze using FOIL. This also focused the experiments on precisely the phenomena we wished to study, as discussed below. Details about the data representation are given in the appendix.

We created a learning task using an attribute on movies — opening-weekend box office receipts. We discretized the attribute so that a positive value indicates a movie with more than $2 million in opening weekend receipts (*prob(+)=0.55*). We call this discretized attribute *receipts* and use it as a binary class label. We also created ten random attributes on studios. The values of these attributes were randomly drawn from a uniform distribution of five values, and thus were independent of the class label. Figure 3 shows the schema with movies, studios, and their attributes.

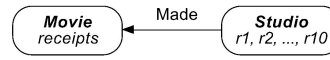


Fig. 3. A simplified data schema, with attributes, for the artificial data sets used for experiments in this section. Attributes denoted *r1* through *r10* are random attributes.

We used simple random partitioning to create (approximately) equal-sized training and test sets. This divides our sample of 1382 movies in two subsamples, each containing approximately 690 movies and their affiliated studios. Each movie appears in

[1] Each of the movies is related to one primary studio. For movies with more than one associated studio, we chose the U.S. studio with highest degree to be the primary, for movies without any U.S. studios we chose the studio of highest degree to be the primary.

only one sample. Each affiliated studio might appear in one or both subsamples, and would never appear more than once in any one subsample. However, a single studio object can be linked to many movies in a given subsample.

Given these data sets, we evaluated the ability of FOIL to learn useful models in the traditional way. We ran FOIL on the training set and evaluated the accuracy of the resulting models on the test set. Given that attributes on studios were created randomly, the expected error for the models constructed exclusively from those attributes should equal the default error (0.55). Deviations from this error represent a bias, which can be measured by subtracting the measured error $\hat{e}$ from the theoretical error e. Positive bias over many trials indicates that the test set accuracy is systematically lower than the theoretical error.

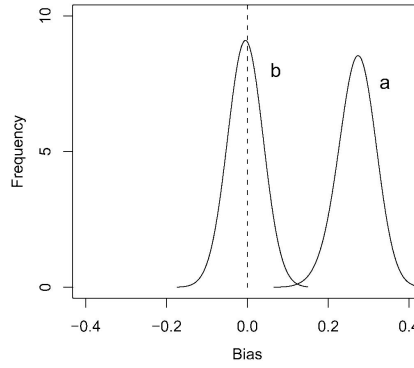


Fig. 4. Distribution of test set bias for FOIL models constructed from random attributes using (a) the actual class labels and (b) randomized class labels. The distributions were smoothed using a bandwidth parameter of 0.4.

Figure 4 shows two distributions of bias estimated from 50 different training and test set partitions. The rightmost distribution (a) results from the experiment described above. The bias is substantially larger than zero, indicating that the measured error of FOIL rules on the test set is much lower than the default. The leftmost distribution (b) results from running the same experiment, except that the values of the class label on movies (*receipts)* are randomly reassigned before each trial. This distribution has almost precisely the expected bias of zero.

These experimental results raise obvious questions: Why does the algorithm appear to learn from random features formed from studios when the actual class label is used, but not when the values of that class label are randomly assigned? What does this result tell us about evaluating relational learners in general?

2 Linkage, Autocorrelation, and Overfitting

Our analysis indicates that biases like that shown in Figure 4 result from the confluence of three common phenomena in relational learning — *concentrated linkage*, *relational autocorrelation*, and *overfitting*. Concentrated linkage and relational autocorrelation create statistical dependence among instances in different samples, and

overfitting exploits that dependence in ways that lead to bias in test set accuracy. We define concentrated linkage and relational autocorrelation formally below. Informally, concentrated linkage occurs when many objects are linked to a common neighbor, and relational autocorrelation occurs when the values of a given attribute are highly uniform among objects that share a common neighbor. Overfitting is familiar to machine learning researchers as the construction of complex models that identify unique characteristics of the training set rather than statistical generalizations present in the population of all data.

Much of the text of this section is drawn from an earlier paper (Jensen & Neville 2002). However, the definitions are so central to understanding the experiments in later sections that we present this material again rather than attempting to summarize.

2.1 Concentrated Linkage

We define concentrated linkage *L(X,P,Y)* with respect to the same conditions as simple random partioning — two sets of objects *X* and *Y* and a set of paths *P* such that *p(x,y)*.

Definition: D_{yX} is the degree of an object *y* with respect to a set of objects *X*. That is, the number of $x \in X$ such that $p(x,y) \in P$. For example, D_{yX} might measure, for a given studio *y*, the number of movies (*X*) it has made. ■

Definition: *Single linkage* of *X* with respect to *Y* occurs in a data set whenever, for all $x \in X$ and $y \in Y$:

$$D_{xY} = 1 \quad \textit{and} \quad D_{yX} \geq 1$$ ■

In these cases, many objects in *X* (e.g., movies) connect to a single object in *Y* (e.g., a studio). We use single linkage as an important special case in future discussions.

Definition: The *concentrated linkage L(x,X,P,Y) of an individual object x* (e.g., a movie) that is linked to objects *Y* (studios) via paths *P* is:

$$L(x, X, P, Y) = \sum_{\substack{y \ s.t. \\ p(x,y) \in P}} \frac{(D_{yX} - 1)}{D_{yX}} \Big/ {D_{xY}}^2$$ ■

the quantity $(D_{yX} - 1)/D_{yX}$ within the summation is zero when the D_{yX} is one, and asymptotically approaches one as degree grows, and thus is a reasonable indicator of *L(x,X,P,Y)*, given single linkage of *x* with respect to *Y*. Because *x* may be linked to multiple nodes in *Y*, we define the average across all nodes y_i linked to *x*, and divide by an additional factor of D_{xY} to rate single linkage more highly than multiple linkage.

Definition: The *concentrated linkage L(X,P,Y) of a set of objects X* (e.g., all movies) that are linked to objects *Y* is:

$$L(X, P, Y) = \sum_{x \in X} \frac{L(x, X, P, Y)}{|X|}$$ ■

Given particular types of linkage, L can be calculated analytically from the sufficient statistics $|X|$ and $|Y|$. For example, in the case of single linkage of X with respect to Y, $L = (|X|-|Y|)/|X|$. For example, the data set shown in Figure 1b exhibits single linkage, so $L(X,P,Y) = 0.60$. Propositional data also display single linkage, and because $|X|=|Y|$, $L(X,P,Y) = 0$. Calculations of several types of linkage are shown for the movie data in Table 1.

Table 1. Linkage in the movie data

Linkage Type	Value
L(Movie, Made, Studio)	0.91
L(Movie, Directed, Director)	0.23
L(Movie, Produced, Producer)	0.08
L(Movie, ActedIn, Actor)	0.01

In addition to the movie data, we have encountered many other instances of concentrated linkage. For example, while studying relationships among publicly traded companies in the banking and chemical industries, we found that nearly every company in both industries uses one of only seven different accounting firms. In work on fraud in mobile phone networks, we found that 800 numbers, 900 numbers, and some public numbers (e.g., 911) produced concentrated linkage among phones. Concentrated linkage is also common in other widely accessible relational data sets. For example, many articles in the scientific literature are published in single journals and many basic research articles are cited in single review articles. On the Web, many content pages are linked to single directory pages on sites such as Yahoo and Google.

2.2 Correlation and Autocorrelation

We will define relational correlation $C(X,f,P,Y,g)$ with respect to two sets of objects X and Y, two attributes f and g on objects in X and Y, respectively, and a set of paths P that connect objects in X and Y.

Definition: *Relational correlation* $C(X,f,P,Y,g)$ is the correlation between all pairs $(f(x),g(y))$ where $x \in X$, $y \in Y$ and $p(x,y) \in P$. ■

Given the pairs of values that these elements define, traditional measures such as information gain, chi-square, and Pearson's contingency coefficient can be used to assess the correlation between values of the attributes f and g on objects connected by paths in P. The range of C depends on the measure of correlation used.

We can use the definition of relational correlation $C(X,f,P,Y,g)$ to define relational *autocorrelation* as the correlation between the same attribute on distinct objects belonging to the same set.

Definition: *Relational autocorrelation C'* is:

$$C'(X,f,P) \equiv C(X,f,P,X,f) \quad \text{where} \quad \forall p(x_i,x_j) \in P \quad x_i \neq x_j$$ ■

For example, C' could be defined with respect to movie objects, the attribute *receipts* on movies, and paths formed by traversing *Made* links that connect the movies to an intervening studio.

If the underlying measure of correlation varies between zero and one, then $C'=1$ indicates that the value of the attribute for a specific node x_i is always equal to all other nodes x_j reachable by a path in P. When $C'=0$, values of $f(X)$ are independent. Table 2 gives estimates of relational autocorrelation for movie receipts, linked through studios, directors, producers, and actors. For a measure of correlation, Table 2 uses Pearson's corrected contingency coefficient (Sachs 1992), a measure that produces an easily interpreted value between zero and one. Autocorrelation is fairly strong for all object types except actors.

In addition to the movie data, we have encountered many other examples of high relational autocorrelation. For example, in our study of publicly traded companies, we found that when persons served as officers or directors of multiple companies, the companies were often in the same industry. Similarly, in biological data on protein interactions we analyzed for the 2001 ACM SIGKDD Cup Competition, the proteins located in the same place in a cell (e.g., mitochondria or cell wall) had highly autocorrelated functions (e.g., transcription or cell growth). Such autocorrelation has been identified in other domains as well. For example, fraud in mobile phone networks has been found to be highly autocorrelated (Cortes et al. 2001). The topics of authoritative web pages are highly autocorrelated when linked through directory pages that serve as "hubs" (Kleinberg 1999). Similarly, the topics of articles in the scientific literature are often highly autocorrelated when linked through review articles.

Table 2. Autocorrelation in the movie data

Autocorrelation Type	Value
C'(Movie,Receipts,Made|Studio|Made)	0.47
C'(Movie,Receipts,Directed|Director|Directed)	0.65
C'(Movie,Receipts,Produced|Producer|Produced)	0.41
C'(Movie,Receipts,ActedIn|Actor|ActedIn)	0.17

Note: The notation $a|x|b$ to denote paths with links of type a and b and intervening objects of type x.

We define relational autocorrelation in a similar way to existing definitions of temporal and spatial autocorrelation (see, for example, Cressie 1993). Autocorrelation in these specialized types of relational data has long been recognized as a source of increased variance. However, the more general types of relational data commonly analyzed by relational learning algorithms pose even more severe challenges because the amount of linkage can be far higher than in temporal or spatial data and because that linkage can vary dramatically among objects.

Relational autocorrelation represents an extremely important type of knowledge about relational data, one that is just beginning to be explored and exploited for learning statistical models of relational data (Neville and Jensen 2000; Slattery and Mitchell 2000). Deterministic models representing the extreme form of relational autocorrelation have been learned for years by ILP systems. By representing and using relational autocorrelation, statistical models can make use of both partially labeled data sets and high-confidence inferences about the class labels of some nodes to increase the confidence with which inferences can be made about nearby nodes.

However, as we show below, relational autocorrelation can also greatly complicate learning of all types of relational models. As we seek to represent and use relational autocorrelation in statistical models of relational data, we will need to adjust for its effects when evaluating more traditional types of features in these models.

3 Effects of Linkage, Autocorrelation, and Overfitting on Bias

The results reported so far for concentrated linkage and relational autocorrelation provide important clues to the behavior shown in Figure 4. Studios are the objects in the movie data that have the highest combination of concentrated linkage and relational autocorrelation. In this section, we show that, if linkage and autocorrelation are both high for a single type of object, and an algorithm produces overfitted models that use attributes on those objects, then the test set error will be biased.

3.1 Dependent Training and Test Sets

Given the definition of simple random partitioning in the introduction, we can examine the statistical dependence of subsamples it produces. We define $prob_{ind}(A,B)$ to be the probability that subsamples A and B are independent[2]. Further, we define $prob_{ind}(A,B|y)$ to be the probability that subsamples A and B are independent with respect to a specific object $y \in Y$, that is where A and B are independent with respect to objects $x_i \in X$ such that $p(x_i,y) \in P$.

Theorem: Given simple random partitioning of a relational data set S with single linkage and $C'=1$:

$$prob_{ind}(A,B) \to 0 \quad as \quad L \to 1.$$

Proof: First, consider a sample S composed of independent subgraphs such as those shown in Figure 1a. In such a sample, $L=0$ because no x is linked to more than one y, and $prob_{ind}(A,B)=1$ because no object x can fall into more than one sample and there is no dependence among objects x_1 and x_2 because $L = 0$.

Now consider the effect of increasing D_{yX}, the degree of an object y (e.g., a studio) with respect to objects X (e.g., movies). For notational convenience, $d=D_{yX}$. Increasing d necessarily increases L for a single object x, because for single linkage $L(x,X,P,Y)=(d-1)/d$. For any one object y, the samples A and B are not independent if both contain an object x_i, such that $p(x_i,y)$. Thus:

$$prob_{ind}(A,B \mid y) = b(0,d,p) + b(d,d,p) \tag{1}$$

where $b(m,n,p)$ denotes the value of the binomial distribution for m successes in n trials, where each trial succeeds with probability p. Here, for example, $b(d,d,p)$ is the probability that a given sample (e.g., A) will contain all of the d movies linked to a given studio y. In the case of simple random partitioning into equal-sized samples, $p=0.5$. For high values of d (many objects x are connected to a single object y), $prob_{ind}(A,B|y)$ approaches zero. For $d=2$, $prob_{ind}(A,B|y)=0.5$. That is, there is only a 50% probability that both objects x connected to y will end up in the same sample. For $d=3$, $prob_{ind}(A,B|y)=0.25$; for $d=10$, $prob_{ind}(A,B|y)=0.002$.

Given that S contains many objects Y, the probability of independence for *all* objects y becomes vanishingly small. Specifically:

---

[2] In this paper, we consider the effects of dependence between instances in different subsamples, but not the effects of dependence among objects within the *same* subsample. The latter topic is covered in another recent paper (Jensen and Neville 2002).

$$prob_{ind}(A,B) = \prod_{y} prob_{ind}(A,B \mid y) \tag{2}$$

For even small samples, $prob_{ind}(A,B)$ goes quickly to zero as L increases. For example, given a sample of 50 instances of X, if $d=2$ $(L=0.5)$, then $prob_{ind}(A,B)=3.0x10^{-8}$. For $d=5$ $(L=0.8)$, $prob_{ind}(A,B)=9.1x10^{-13}$. For large samples and L>0, $prob_{ind}(A,B) \approx 0$. ν

Not only is the probability of *any* dependence between A and B high, but the degree of dependence is very likely to be high. For example, the binomial distribution can be used to derive the expected number of objects x_1 in sample A with a matching object x_2 in B such that $p(x_1,y) \in P$ and $p(x_2,y) \in P$ for some $y \in Y$ with degree $d=D_{yX}$. The maximum and expected number of matched pairs of dependent instances is:

$$Max(pairs) = \lfloor d/2 \rfloor \tag{3}$$

$$E(pairs) = \sum_{i=1}^{d} \min(i, d-i) b(i, d, 0.5)$$

For example, for $d=5$, the maximum number of pairs is $Max(pairs)=2$ and the expected number is $E(pairs)=1.56$. For $d=10$, $Max(pairs)=5$ and $E(pairs)=3.77$.

3.2 How Bias Varies with Autocorrelation, Linkage, and Overfitting

Given dependent training and test sets, it is relatively easy to see how overfitted models can cause bias in test set accuracy. One way of characterizing the observed behavior is that it represents a relational version of the "small disjuncts" problem (Holte, Acker, & Porter 1989). This problem arises in propositional learning when overfitted models parse the instance space into sets ("disjuncts") containing only a few data instances. For example, when a decision tree is pathologically large, its leaf nodes can apply to only a single instance in the training set. Such models perform as lookup tables that map single instances to class labels. They achieve very high accuracy on training sets, but they generalize poorly to test sets, when those test sets are statistically independent of the training set.

In the relational version of the small disjuncts problem, models use relational attributes to parse the space of objects y into sets so small that they can uniquely identify one such object (e.g., a single studio). If that object is linked to many objects x where a single class predominates (e.g., *receipts* = +), then a model that uniquely identifies that object y can perform well on the training data. If that y also appears in the test data, then the model can perform well on test data.

Indeed, such overfitting is made more likely by high autocorrelation among the values of the class label *within* a given training set. If several objects X in a training set are all linked to a single object y, and if the class labels of the objects X are highly correlated, then it is more likely that a learning algorithm will create a model with components intended to predict precisely these instances than if only a single instance had this combination of attribute values and class label.

As noted above, relational autocorrelation represents an extremely important type of knowledge about relational data, one that can be exploited to improve accuracy (Neville and Jensen 2000; Slattery and Mitchell 2000). However, it can also fool algorithms and evaluation techniques not designed to account for its effects.

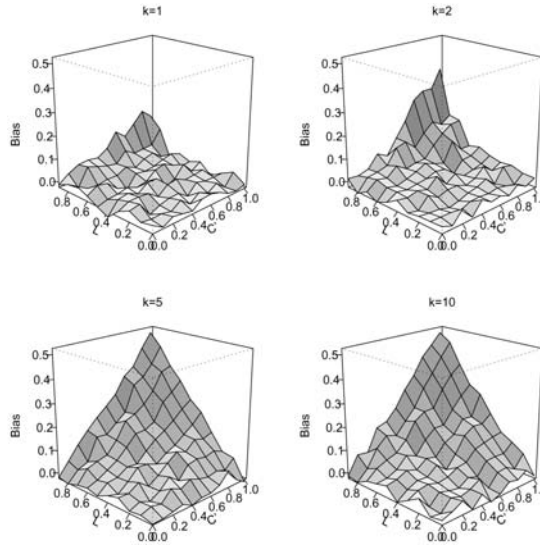


Fig. 5. Bias increases with linkage (*L*), autocorrelation (*C'*), and number of random attributes (*k*). Each point represents the average of 20 trials.

For example, consider the results shown in Figure 5. The graphs show how the bias varies across a wide range of linkage (*L*), autocorrelation (*C'*), and potential for overfitting. To alter this latter characteristic of learning algorithms, we varied the number of attributes *k* from one to ten. In each trial, we created synthetic data sets with 200 objects *X* and specified values of autocorrelation and single linkage with objects *Y*. Each object *x* was given a class label drawn from a binary uniform distribution. Each object *y* was given *k* attributes, each with a value drawn from a five-valued uniform distribution. This sample was then divided into equal-sized training and test sets using simple random partitioning. We applied FOIL to the training set, and then evaluated the error of the resulting rules on the test set. For each combination of *L*, *C'*, and *k*, we ran 20 trials and averaged the bias.

For a given number of attributes *k*, bias increases dramatically with increasing linkage *L* and autocorrelation *C'*. For high values of *k*, *L*, and *C'*, bias is maximal (0.5). However, even moderate values of *k*, *L*, and *C'* produce substantial bias, confirming the results in the first section.

It is important to note that in the experiments presented above, these overfitted models are *not* learning any general knowledge about autocorrelation and linkage. For example, the FOIL rules learned for the experimental results depicted in Figure 4 contain only clauses of the form:

```
receipts(A) :- made-by(A,B), studio-attributes(B,C,D,E,F).
```

That is, they exclusively relate attributes of studios to *receipts* of movies linked to them. As noted previously, linkage and autocorrelation represent an important type of knowledge that could be exploited by a relational learning algorithm. However, most relational learning algorithms either cannot or do not learn probabilistic models of this form. The rules formed by FOIL on the synthetic data used in Figure 5 are of a similar

form. The results in Figure 5 show that large bias that can result when algorithms learn overfitted models from data with strong linkage and autocorrelation.

4 Subgraph Sampling

Fortunately, this bias can be eliminated by a relatively small change to the procedure for creating training and test sets. *Subgraph sampling* guarantees that an object y and corresponding objects X appear only within a single subsample. This confines any autocorrelation among the class labels of objects X to a single subsample, and thus removes the dependence between subsamples due to concentrated linkage and relational autocorrelation.

We first introduced subgraph sampling of relational data in an earlier paper (Jensen and Neville 2001). However, we lacked a full understanding of the causes of dependence between subsamples, and we proposed an extreme form of subgraph sampling that eliminated all possible duplication of objects between subsamples, even when class labels were not autocorrelated through all types of objects. Here we propose a form of subgraph sampling that is far more conservative.

First, consider the special case where linkage and autocorrelation are high for only one type of object y, and that object exhibits single linkage with objects X. For example, in the movie data, only studios exhibit both high linkage and high autocorrelation; other types of objects (actors, directors, and producers) have fairly low values for one or both quantities. In addition, studios exhibit single linkage with movies. In this special case, we can partition a sample S based on the objects Y (e.g., studios), and then place all objects X in the same subsample as their corresponding y.

A more general partitioning algorithm first assigns objects X to prospective samples, and then incrementally converts prospective assignments to permanent assignments only if the corresponding objects Y for the given x are disjoint from other objects Y already assigned to subsamples other than the prospective subsample of x. In contrast to the approach we proposed earlier (Jensen and Neville 2001), the objects Y considered during sampling should only be those through which linkage and autocorrelation is high[3].

One feature of the general algorithm is worthy of special note — the random assignment of objects X to "prospective" subsamples. The algorithm either makes a prospective assignment permanent, or discards the object. An alternative algorithm would search for an assignment of objects to permanent subsamples that maximizes the number of objects assigned to each subsample, thus maximizing the size of subsamples. However, this approach can induce another form of statistical dependence among subsamples. Consider how such an "optimizing" algorithm would behave when confronted with a data set consisting of two disjoint (or nearly disjoint) sets of relational data. One subsample would be filled entirely with objects from one disjoint set, and another would be filled with objects from the other set. If the statistical characteristics of one of the disjoint sets did not mirror the characteristics of the other, then accuracy estimates of learned models would be biased downward.

[3] What is considered "high" would vary somewhat by the desired precision of the estimated accuracy. This is a topic for future work.

Subgraph sampling resembles techniques that construct samples from a small number of completely disconnected graphs. For example, some experiments with WEBKB (Slattery and Mitchell 2000) train classification models on pages completely contained within a single website, and then test those models on pages from another website with no links to the training set. This approach exploits a feature of some websites — heavy internal linkage but few external links. Similarly, some work in ILP constructs samples from sets of completely disconnected graphs (e.g., individual molecules or English sentences) (Muggleton 1992). This approach are possible only when the domain provides extremely strong natural divisions in the graphs, and this approach is only advisable where the same underlying process generated each graph. In contrast, subgraph sampling can be applied to data without natural divisions. Where they exist, subgraph sampling will exploit some types of natural divisions. Where they do not exist, logical divisions can be created that preserve the statistical independence among samples.

5 Subgraph Sampling Eliminates Bias

In this section, we show how subgraph sampling eliminates the bias caused by linkage, autocorrelation, and overfitting. First, we replicate the experiments that produced Figure 4. However, rather than learning models for a randomized class label, we learn models on the original class label, but with samples produced by subgraph sampling. The results are shown in Figure 6. As before, the bias associated with simple random partitioning is high. However, the distribution of bias for subgraph sampling (b) has a mean bias near zero.

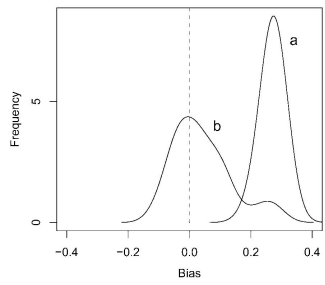


Fig. 6. Bias of FOIL models using random attributes for (a) simple random partitioning and (b) subgraph sampling.

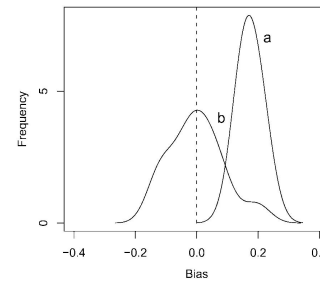


Fig. 7. Bias in FOIL models using non-random attributes for (a) simple random partitioning and (b) subgraph sampling.

The results in figures 4 and 6 were obtained using completely random attributes artificially generated on studios. However, similar results are obtained if we learn from attributes generated from the real characteristics of studios. The results in Figure 7 were generated by learning models with four attributes on studios. The attributes are the first letter of the studio name, the decade in which the studio was founded, the number of letters in the studio name (discretized to 10 unique values), and a binary attribute indicating whether the studio is located in the U.S. As before, the bias is high for simple random partitioning. Bias for both distributions used the mean of the dis-

tribution for subgraph sampling as an estimate of true error. These results confirm that the bias does not result from some peculiarity in the generation of random attributes, but rather results from dependence between the training and test sets.

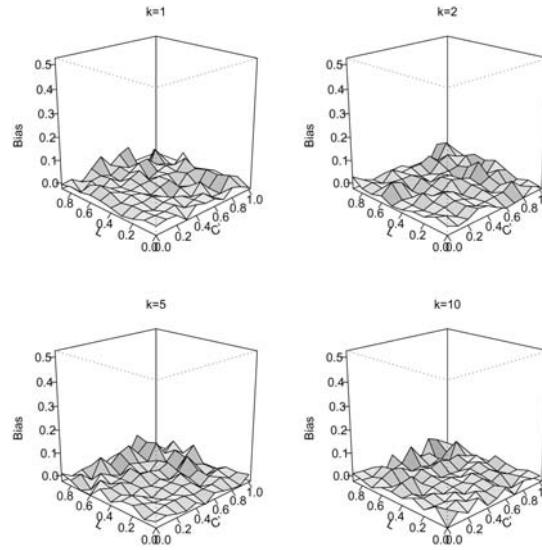


Fig. 8. Subgraph sampling with maximal separation eliminates bias at all levels of linkage (*L*), autocorrelation (*C'*), and number of random attributes (*k*).

These results only indicate bias for a single combination of *L*, *C'*, and degree of overfitting. Figure 8 shows the results of more systematic variation of these quantities. The figure was produced from the same experiments as Figure 5, except the training and test sets were constructed by subgraph sampling rather than simple random partitioning. The result is extremely low bias across the full range of values of *L*, *C'*, and *k*..

6 Conclusions and Future Work

Concentrated linkage and relational autocorrelation can cause strong bias in the test set accuracy of induced models. In this paper, we demonstrate the bias using FOIL, so that other researchers can easily replicate and extend our experiments, but we have also observed this phenomenon in our own algorithms for relational learning. Fortunately, the bias associated with linkage and autocorrelation can be corrected by using subgraph sampling in preference to simple random partitioning.

While some special classes of relational data naturally allow subgraph sampling, relational learning methods will increasingly encounter data in which this bias arises, as we extend our work to more general classes of relational data, including networks of web pages, bibliographic citations, financial transactions, messages, biochemical interactions, computers, supervisory relationships, and social interactions.

This work also emphasizes the need to pursue research on relational learning techniques that exploit relational autocorrelation to enhance the predictive power of rela-

tional models. Additional work should also investigate methods to estimate the bias associated with specific levels of autocorrelation and linkage, and to search for classes of objects that exhibit those degrees of linkage and autocorrelation, so more automated approaches to subgraph sampling can be devised.

Acknowledgments

Foster Provost and Hannah Blau provided valuable comments on an earlier version of this work and Ross Fairgrieve prepared the movie data for analysis. The data used in this paper were drawn from the Internet Movie Database (www.imdb.com) and several experiments used FOIL, by Ross Quinlan. This research is supported by The Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory (AFRL), Air Force Material Command, USAF, under agreements F30602-00-2-0597 and F30602-01-2-0566.

References

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

Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., & Slattery, S. (1998). Learning to extract symbolic knowledge from the World Wide Web. *Proceedings of the Fifteenth National Conference on Artificial Intellligence (*pp. 509-516). Menlo Park: AAAI Press.

Cressie, N. (1993). *Statistics for Spatial Data.* Wiley.

Flach, P. and N. Lachiche (1999). 1BC: a first-order Bayesian classifier. *ILP'99*. 92-103. Springer.

Getoor, L., N. Friedman, D. Koller, and A. Pfeffer (1999). Learning probabilistic relational models. In *IJCAI'99*. 1300-1309.

Holte, R., Acker, L., & Porter, B. (1989). Concept learning and the accuracy of small disjuncts. *Proceedings of the 11th International Joint Conference on Artificial Intelligence* (pp. 813-818). Detroit: Morgan Kaufmann.

Jensen, D. and P. Cohen (2000). Multiple comparisons in induction algorithms. *Machine Learning* 38:309-338.

Jensen, D. and J. Neville (2001). Correlation and sampling in relational data mining. *Proceedings of the 33rd Symposium on the Interface of Computing Science and Statistics.*

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

John, G., R. Kohavi, and K. Pfleger (1994). Irrelevant features and the subset selection problem. *ICML'94*. 121-129.

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

Lipton, R., Naughton, J., Schneider, D., & Seshadri, S. (1993). Efficient sampling strategies for relational database operations. *Theoretical Computer Science*, 116, 195-226.

Muggleton, S. (Ed) (1992). *Inductive Logic Programming.* Academic Press

Muggleton, S. (2000). Learning Stochastic Logic Programs. AAAI Workshop on Learning Statistical Models from Relational Data, 36-41.

Noreen, E. (1989). *Computer Intensive Methods for Testing Hypotheses.* Wiley.

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

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

Sachs, L. (1982). *Applied Statistics.* Springer-Verlag.

Slattery, S. & Mitchell, T. (2000). Discovering test set regularities in relational domains. *Proceedings of the 17th International Conference on Machine Learning.* San Francisco: Morgan Kaufmann.

Appendix

The movie data were coded for input to FOIL as follows: Each studio attribute was specified as an unordered discrete type with all attribute values flagged as a theory constants. Unordered type specifications also define movies and studios, with 1382 unique labels for the movies and 128 unique labels for the studios respectively.

The input contains one target relation and two background relations:

```
receipts(movie)
made-by(studio)
studio-attributes(studio, first-char, name-len, in-us, decade)
```

The target relation described above for movie receipts contains both positive and negative examples. The two background relations contain only positive examples; one specifying the relationships between movies and studios, and the other specifying attribute values associated with each studio.

Learned clauses were similar in form to:

```
receipts(A) :- made-by(A,B), studio-attributes(B,C,D,E,F).
```

All experiments used the current version of FOIL (foil6.sh) obtained from: <http://www.cse.unsw.edu.au/~quinlan/>. Arguments to FOIL specified that negative literals were not to be considered and the minimum accuracy of any clause considered was at least 70%. Other than these two modifications, FOIL's default settings were used.

Learnability of Description Logic Programs

Jörg-Uwe Kietz

kdlabs AG, Flurstr. 32, 8022 Zürich, Switzerland
juk@kdlabs.com
http://www.kdlabs.com/

Abstract. CARIN-$\mathcal{ALN}$ is an interesting new rule learning bias for ILP. By allowing description logic terms as predicates of literals in datalog rules, it extends the normal bias used in ILP as it allows the use of all quantified variables in the body of a clause. It also has at-least and at-most restrictions to access the amount of indeterminism of relations. From a complexity point of view CARIN-$\mathcal{ALN}$ allows to handle the difficult indeterminate relations efficiently by abstracting them into determinate aggregations. This paper describes a method which enables the embedding of CARIN-$\mathcal{ALN}$ rule subsumption and learning into datalog rule subsumption and learning with numerical constraints. On the theoretical side, this allows us to transfer the learnability results known for ILP to CARIN-$\mathcal{ALN}$ rules. On the practical side, this gives us a preprocessing method, which enables ILP systems to learn CARIN-$\mathcal{ALN}$ rules just by transforming the data to be analyzed. We show, that this is not only a theoretical result in a first experiment: learning CARIN-$\mathcal{ALN}$ rules from a standard ILP dataset.

1 Introduction

CARIN was proposed by [Levy and Rouset, 1998] as a combination of the two main approaches to represent and reason about relational knowledge, namely description logic (DL) and first-order horn-logic (HL). In Inductive Logic Programming (ILP) learning first-order horn-logic is investigated in depth, for learning DLs there exist first approaches and theoretical learnability results [Kietz and Morik, 1994; Cohen and Hirsh, 1994; Frazier and Pitt, 1994]. Recently, it was proposed to use CARIN-$\mathcal{ALN}$ as a framework for learning [Rouveirol and Ventos, 2000]. This is an interesting extension of ILP as $\mathcal{ALN}$ provides a new bias orthogonal to the one used in ILP, i.e. it allows all quantified descriptions of body-variables, instead of the existential quantified ones in ILP. This allows to handle the difficult indeterminate relations efficiently by abstracting them into a determinate summary. It also has at-least and at-most restrictions, which allow to quantify the amount of indeterminism of these relations. However, up to now there are neither practical nor theoretical results concerning learning the CARIN-$\mathcal{ALN}$ language.

This paper is intended to close this gap, by showing how CARIN-$\mathcal{ALN}$ learning can be embedded into first-order horn-logic learning as done by ILP-methods. Even, if DL and HL have been shown to be quite incomparable concerning their

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 117–132, 2003.

expressive power [Borgida, 1996] with respect to their usual semantic interpretation of primitive concepts and roles, we show that reasoning in DL can be simulated by reasoning in horn logic with simple numeric constraints as formalized in [Sebag and Rouveirol, 1996] and as present in most ILP-systems, e.g. Foil [Quinlan and Cameron-Jones, 1993] or Progol [Muggleton, 1995]. A simple invertible function encodes normalized concept descriptions into horn clauses using new predicates with an external semantics (as Borgida has shown they are not expressible in horn logic) to represent the DL terms, i.e. from an ILP point of view, learning CARIN-$\mathcal{ALN}$ can be done by learning HL with extended background knowledge that encodes the description logic terms. This encoding not only provides another method to do deductive reasoning, i.e. subsumption, equivalence and satisfiability checking, but also allows ILP methods to learn $\mathcal{ALN}$ concept descriptions and CARIN-$\mathcal{ALN}$ rules. The known border-line of polynomial learnability for ILP can be transfered to CARIN-$\mathcal{ALN}$ as well, as this encoding is a prediction preserving reduction as used in [Cohen, 1995] to obtain PAC-learnability results for ILP.

In section 2 we repeat the basic definitions of the description logic $\mathcal{ALN}$ and define the basis of our new encoding into horn logic. In section 3 we define the description logic program (DLP) formalism. We show how it is related to DLP (3.1) and ILP (3.2) and we define a reasoning procedure by extending the encoding from section 2 to DLP rules (3.3). In section 3.4, we characterize the boarder line of polynomial learnability of DLP rules using the encoding into horn logic. Finally in section 4, we demonstrate with a first experiment, that this encoding can be used to learn DLP rules using a normal ILP-systems on an extended, i.e. preprocessed data-set.

2 The Description Logic $\mathcal{ALN}$

Starting with KL-ONE [Brachman and Schmolze, 1985] an increasing effort has been spent in the development of formal knowledge representation languages to express knowledge about concepts and concept hierarchies. The basic building blocks of description logics are concepts, roles and individuals. Concepts describe the common properties of a collection of individuals and can be considered as unary predicates which are interpreted as sets of objects. Roles are interpreted as binary relations between objects. Each description logic defines also a number of language constructs that can be used to define new concepts and roles. In this paper we use the very basic[1] description logic $\mathcal{ALN}$ under its normal open-world (OWA) semantics, with the language constructs in table 1.

Definition 1 ($\mathcal{ALN}$-terms and their interpretation).
Let P denote a primitive concept, i.e. an unary predicate, and R denote a primitive role, i.e. a binary predicate, n is a positive integer and the C_i are concept terms. The set of all $\mathcal{ALN}$-concept-terms ($\mathcal{C}$) consist of everything in the left

[1] See http://www-db.research.bell-labs.com/user/pfps/papers/krss-spec.ps for further language constructs considered in description logics.

Table 1. Concept terms in $\mathcal{ALN}$ and their model-theoretic interpretation

Term (Math)	Term (Classic)	Interpretation
$\top$	**everything**	Δ^I
$\bot$	**nothing**	$\emptyset$
P	P	P^I
$\neg P$	**not** P	$\Delta^I \setminus P^I$
$C_1 \sqcap \ldots \sqcap C_n$	**and** $C_1 \ldots C_n$	$C_1^I \cap \ldots \cap C_n^I$
$\forall R.C$	**all** R C	$\{x \in \Delta^I \mid \forall y \in \Delta^I :< x, y >\in R^I \Rightarrow y \in C^I\}$
$\geq nR$	**at-least** n R	$\{x \in \Delta^I \mid \ \|\{y \in \Delta^I \mid< x, y >\in R^I\}\| \geq n\}$
$\leq nR$	**at-most** n R	$\{x \in \Delta^I \mid \ \|\{y \in \Delta^I \mid< x, y >\in R^I\}\| \leq n\}$

column of the table 1 (and nothing more) and their interpretation (I, Δ) *(see def. 6 below) with respect to the interpretation of primitive concept and roles is defined in the right row of the table 1 (*C^I *is used as a shortcut for* $I(C)$*, if* C *is not primitive). The set of all* $\mathcal{ALN}$*-role-terms (*$\mathcal{R}$*) is just the set of all primitive roles* R*.*

These language constructs can be used to build complex terms, e.g. the term *train* $\sqcap \forall$ *has_car.*(*car* $\sqcap \leq 0$ *has_load*) can be used to define empty trains (all cars do not have a load), in Michalski's well-known train domain. A statement **not** expressible as a logic program, if we are restricted to the predicates already present in the train domain and cannot use externally computed predicates.

The main reasoning tasks with description logic terms are subsumption, equivalence and satisfiability checking for deduction and the least common subsumer (lcs) for learning.

Definition 2 (subsumption, equivalence, satisfiability and least common subsumer). *The concept description* D *subsumes the concept description* C *(*$C \sqsubseteq D$*) iff* $C^I \subseteq D^I$ *for all interpretations* I*;* C *is satisfiable if there exists an interpretation* I *such that* $C^I \neq \emptyset$*;* C *and* D *are equivalent (* $C \equiv D$*) iff* $C \sqsubseteq D$ *and* $D \sqsubseteq C$*; and* E *is the least common subsumer (lcs) of* C *and* D*, iff* $C \sqsubseteq E$ *and* $D \sqsubseteq E$ *and if there is an* E' *with* $C \sqsubseteq E'$ *and* $D \sqsubseteq E'$*, then* $E \sqsubseteq E'$*.*

We have chosen $\mathcal{ALN}$, because subsumption checking between $\mathcal{ALN}$ descriptions is polynomial [Donini et al., 1991], and in this paper we always consider an empty terminological component since subsumption checking between $\mathcal{ALN}$ terms with respect to an acyclic terminological component is coNP-complete [Nebel, 1990b] A polynomial time algorithm for the lcs computation of $\mathcal{ALN}$ concepts can be found in [Cohen et al., 1992].

$\mathcal{ALN}$ descriptions are in general not normalized, i.e. there exist several possibilities to express the same concept, e.g. $\bot \equiv (A \sqcap \neg A) \equiv (\geq 2\ R \sqcap \leq 1\ R)$. However, they can be normalized, e.g. by the following set of rewrite rules.

Definition 3 (Normalization of $\mathcal{ALN}$ descriptions). *norm*$(C) = C'$ *iff sorted*$(C) \mapsto \ldots \mapsto C'$ *and no rewrite rule is applicable to* C'*, i.e as a first step any conjunction* $(C_1 \sqcap \ldots \sqcap C_n)$ *is replaced by its sorted equivalent for a total order* $<$ *that respects* $(P_1) < (\neg P_1) < \ldots < (P_n) < (\neg P_n) < (\geq n_1\ R_1) <$

$(\leq m_1\ R_1) < (\forall R_1.C_1) < ... < (\geq n_n\ R_n) < (\leq m_n\ R_n) < (\forall R_n.C_n)$. *Then the following rewrite rules are applied to all conjunctions not only the top-level one, as long as possible.*

1. $(C_1 \sqcap \ldots \sqcap C_n) \mapsto \bot$, if any $C_i = \bot$
2. $(C_1 \sqcap \ldots \sqcap C_n) \mapsto C_1 \sqcap \ldots \sqcap C_{i-1} \sqcap C_{i+1} \sqcap \ldots \sqcap C_n$, if any $C_i = \top$
3. $(P \sqcap \neg P) \mapsto \bot$
4. $(\leq n\ R \sqcap \geq m\ R) \mapsto \bot$, if $n < m$.
5. $(C \sqcap C) \mapsto C$
6. $(\geq 0 R) \mapsto \top$
7. $(\geq n_1\ R \sqcap \geq n_2\ R) \mapsto \geq maximum(n_1, n_2)\ R$
8. $(\leq n_1\ R \sqcap \leq n_2\ R) \mapsto \leq minimum(n_1, n_2)\ R$
9. $(\leq 0\ R \sqcap \forall R.C) \mapsto \leq 0\ R$
10. $(\forall R.\bot) \mapsto \leq 0\ R$
11. $(\forall R.\top) \mapsto \top$
12. $(\forall R.C_1 \sqcap \forall R.C_2) \mapsto \forall R.(merge^2(C_1 \sqcap C_2))$

Lemma 1. *For any two $\mathcal{ALN}$ concept descriptions C_1 and C_2: equivalence ($C_1 \equiv C_2$, iff $norm(C_1) = norm(C_2)$), subsumption ($C \sqsubseteq D$, iff $norm(C \sqcap D) = norm(C)$) and satisfiability ($C$ is satisfiable, iff $norm(C) \neq \bot$) can be computed in $O(n \log n)$, with n being the size of $C \sqcap D$. Proof in [Kietz, 2002]*

The main innovation of this paper is the following encoding of description logic terms into horn clauses. From the logical point of view the new predicates are primitive as all predicates, i.e. they have an external semantic chosen freely by the interpretation function as for any other predicate. The important matter is, that we can prove, that they are always **used** such that semantic constraints (e.g. $I\langle X/\{a_1, \ldots, a_n\}\rangle(cp_P(X)) = 1$, iff $\{a_1, \ldots, a_n\} \subseteq I(P)$) on the interpretation that would formalize the semantic of the encoded DL-terms are respected, i.e that this function is indeed a correct polynomial problem (prediction and deduction preserving) reduction of IDLP to ILP.

Definition 4 ($\mathcal{ALN}$ encoding into constraint horn logic).
$\Phi(C) = h(X) \leftarrow \Phi(norm(C), X)$.
$\Phi(\bot, X) = \bot(X)$
$\Phi(P\ \{\sqcap\ C\}, X) = cp_P(X)\{, \Phi(C, X)\}$
$\Phi(\neg P\ \{\sqcap\ C\}, X) = cn_P(X)\{, \Phi(C, X)\}$
$\Phi(\geq nR \sqcap \leq mR \sqcap \forall R.C_R\ \{\sqcap\ C\}, X) = rr_R(X, [n..m], Y), \Phi(C_R, Y)\{, \Phi(C, X)\}$
$\Phi(\leq mR \sqcap \forall R.C_R\ \{\sqcap\ C\}, X) = rr_R(X, [0..m], Y), \Phi(C_R, Y)\{, \Phi(C, X)\}$
$\Phi(\geq nR \sqcap \forall R.C_R\ \{\sqcap\ C\}, X) = rr_R(X, [n..*], Y), \Phi(C_R, Y)\{, \Phi(C, X)\}$
$\Phi(\geq nR \sqcap \leq mR\ \{\sqcap\ C\}, X) = rr_R(X, [n..m], Y)\{, \bot(Y)\ \textit{if}\ m = 0\}\{, \Phi(C, X)\}$
$\Phi(\forall R.C_R\ \{\sqcap\ C\}, X) = rr_R(X, [0..*], Y), \Phi(C_R, Y)\{, \Phi(C, X)\}$
$\Phi(\leq mR\ \{\sqcap\ C\}, X) = rr_R(X, [0..m], Y)\{, \bot(Y)\ \textit{if}\ m = 0\}\{, \Phi(C, X)\}$
$\Phi(\geq nR\ \{\sqcap\ C\}, X) = rr_R(X, [n..*], Y)\{, \Phi(C, X)\}$

[2] Merging two sorted lists into one sorted list as in merge-sort. In an optimized implementation the recursive application of the normalisation rules should be integrated into that linear (in the size of the conjunctions) process.

where Y is always a new variable not used so far and $\{\sqcap C\}$ means, if there are conjuncts left, recursion on them $\{, \Phi(C, X)\}$ has to continue. For any normalised concept term only the first matching one has to by applied.

Let us illustrate our encoding function on our train example:
$\Phi(train \sqcap \forall\ has\_car.(car \sqcap \leq 0\ has\_load)) = h(X) \leftarrow cp\_train(X)$, $rr\_has\_car(X, [0..*], Y)$, $cp\_car(Y)$, $rr\_has\_load(Y, [0..0], Z)$, $\bot(Z)$. Note, that this clause has a very different meaning than the clause $h(X) \leftarrow train(X)$, $has\_car(X, Y)$, $car(Y)$, $has\_load(Y, Z)$. The first one must be interpreted as being true for every **set of empty trains**, i.e. X, Y and Z are variables over set of terms. The second one is true for every **single train with has a car, which has a load**. The predicates (and variables) in the first clauses are meta-predicates (set-variables) over the ones in the second clause (individual variables), e.g. like `findall` in Prolog with a specific call using predicates of the second clause. This difference becomes especially important in our DLP language, i.e. in the next section, where both kinds of literals can occur in a single clause.

Nevertheless, we are now nearly able to simulate DL subsumption with θ-subsumption and lcs with lgg. There are only two very small and easy extensions of θ-subsumption and lgg (to $\theta_{I\bot}$-subsumption and $lgg_{I\bot}$) needed:

The handling of sub-terms representing intervals of numbers, e.g. a term like $[0..*]$ should θ_I-subsume a term like $[1..5]$. More precisely an interval $[Min_1..Max_1]$ θ_I-subsumes an interval $[Min_2..Max_2]$, iff $Min_1 \leq Min_2$ and $Max_2 \leq Max_1$; and the lgg_I of two intervals $[Min_1..Max_1]$ and $[Min_2..Max_2]$ is the interval $[minimum(Min_1, Min_2)..maximum(Max_1, Max_2)]$ [3]. The handling of nothing, i.e. $\bot(X)$ should be $\theta_\bot$-subsumed by $\Phi(C, X)$ for any concept description C, e.g. by any sub-clause containing the relation-chains starting with X; and the $lgg_\bot$ of $\bot(X)$ and $\Phi(C, X)$ is $\Phi(C, X)$. This does not really need an extension of the logic, it can be simulated with normal θ-subsumption and the following sub-clause for bottom: for any primitive role R: $rr_R(bottom, [*..0], bottom)$ and for any primitive concept C: $cp_C(bottom)$, $cn_c(bottom)$. Every $\Phi(C, X)$ θ-subsumes this cyclic structure with a θ that maps every variable in it to the term *bottom* and the lgg of every $\Phi(C, X)$ and that sub-clause is $\Phi(C, X)$, e.g. $lgg_I(rr_R(bottom, [*..0], bottom),\ rr_R(X, [Min..Max], Y)) = rr_R(X, [Min..Max], Y)$ and everything present in $\Phi(C, X)$ has a selection [Plotkin, 1970] within the lgg.

Theorem 1 (Simulation of subsumption and lcs). *A concept description C subsumes a concept description D ($D \sqsubseteq C$), if and only if the encoding*

[3] Formally this corresponds to reasoning and learning with simple numeric constraints as present in Constraint Logic Programs (CLP). In in most ILP-systems, e.g. Foil [Quinlan and Cameron-Jones, 1993] or Progol [Muggleton, 1995] this is done with the help of the computed built-in predicates $\leq$ and/or $\geq$. For Foil or Progol a more suitable encoding is $rr_R(X, n_{atleast}, m_{atmost}, Y)$, as they can learn c_{min} and c_{max} in literals like $c_{min} \leq n_{atleast}$ and $m_{atmost} \leq c_{max}$. [Sebag and Rouveirol, 1996] have analysed this CLP extension of ILP systems and it is easy to see that the properties of θ-subsumption also hold for θ_I-subsumption.

of C $\theta_{I\perp}$-subsumes the encoding of D ($\Phi(C) \vdash_{\theta_{I\perp}} \Phi(D)$), and $lcs(C,D) \equiv \Phi^{-1}(lgg_{I\perp}(\Phi(C),\Phi(D)))$

This theorem directly follows from the similarity between the encoding Φ of the normalized concept terms and $\theta_{I\perp}$-subsumption and the correctness of the structural subsumption algorithm given in [Cohen et al., 1992]. There are some more nice properties for learning. Any clause which subsumes a set of such clauses, does so deterministically [Kietz and Lübbe, 1994], i.e. is determinate [Muggleton and Feng, 1992], as any $rr_R(X,I,Y)$ occurs just once for any variable X. The relation chains in the clause (i.e. the chains of $rr_R(X,I,Y)$ literals) are not only acyclic but even tree-structured (as the DL-Term is a tree), i.e. subsumption (coverage test), learning, and propositionalisation are very easy, e.g. the depth limit i needed for the polynomial learnability of (cyclic) ij-determinate clauses is not needed. Φ is a bijective, invertible function, i.e $\Phi^{-1}(\Phi(C)) \equiv C$, i.e. it allows to decode generalized (i.e. learned) clauses: If D is a linked clause[4], and $D \vdash_{\theta_{I\perp}} \Phi(C)$ for any $\mathcal{ALN}$ description C, then $\Phi^{-1}(D)$ is totally invertible and produces a valid description logic term.

3 Induction of Description Logic Programs

CARIN as proposed in [Levy and Rouset, 1998] combines first-order function-free horn logic with description logic by allowing description logic terms as body literals in horn rules. Concept terms represent unary predicates and role-terms represent binary predicates. The direct use of primitive concepts and roles is indistinguishable from the use of unary and binary predicates in FOL, but the use of concept terms, i.e. descriptions contain all, at-least and at-most adds expressive power to the language. Here is an example of a CARIN-$\mathcal{ALN}$ rule using this expressive power. Note, that this cannot be expressed in neither horn logic nor $\mathcal{ALN}$ alone. We bracket DL-terms with [], wherever we think that this will increase readability.

$$east\_train(X) \leftarrow has\_car(X,Y),\ has\_car(X,Z), same\_shape(Y,Z),\\ [train \sqcap \leq 2\ has\_car \sqcap \forall\ has\_car.[car \sqcap \leq 0\ has\_load]](X).$$

The definition of our DLP language is based on datalog, i.e. function-free logic programs. The important extension is that we allow $\mathcal{ALN}$-concept-terms as one-place predicates and $\mathcal{ALN}$-role-terms as two-place-predicates.

Definition 5 (The DLP language). *Variables $\mathcal{V}$, constants $\mathcal{F}$, and terms $\mathcal{T}$ are as usual in datalog. Let $\mathcal{P} = (\mathcal{P}_n)_{n\in\mathbb{N}}$ be a signature of predicate symbols, i.e. for $n \in \mathbb{N}$ is $p \in \mathcal{P}_n$ a predicatssymbols of arity n, e.g. $p,q,r \in \mathcal{P}_n$. The extension to description logic programms extends $\mathcal{P}_1$ by the set of all concept-terms $\mathcal{C}$ and $\mathcal{P}_2$ by the set of all role-terms $\mathcal{R}$ (see Definition 1). Basesd on that atoms AT, literals LIT, clauses, facts, rules and desricption logic programs are again defined as in datalog.*

[4] There is no partition of literals such that one partition does not share at-least one variable with any other partition.

Definition 6 (Interpretations and Models of clause sets and description logic programs).
Let Δ be a domain and I an interpretation function mapping clauses to truth values (0 and 1), n-ary predicates ($P_n \in \mathcal{P}$) to n-ary relations (written P_n^I) over the domain Δ, terms ($t \in \mathcal{F}$) to elements of Δ (written t^I), $\mathcal{ALN}$-concept-terms ($C \in \mathcal{C}$)as defined in def. 1 and $\mathcal{ALN}$-role-terms ($R \in \mathcal{R}$)as defined in def. 1. An interpretation (I, Δ) satisfies (is a model of) a clause set, iff every clause is satisfied by (I, Δ). A clause is satisfied by (I, Δ), iff the interpretation function assigns the truth value 1 to it. For all interpretations (I, Δ) the following must hold[5]*:*

1. $\forall t_1, t_2 \in \mathcal{F} : t_1 \neq t_2$ *iff* $t_1^I \neq t_2^I$ [6],
2. $I(\Box) = 0$, *and* $\forall X \in \mathcal{T} : I(\bot(X)) = 0$
3. $I(\{l_1, \ldots, l_n\}) = 1$, *iff for all vectors* $\boldsymbol{a} = a_1, \ldots, a_m$, $a_i \in \Delta, (1 \leq i \leq m) : I\langle X_1/a_1, \ldots, X_m/a_m\rangle(l_1) = 1$ *or* ... *or* $I\langle X_1/a_1, \ldots, X_m/a_m\rangle(l_n) = 1$, *where* $\boldsymbol{X} = X_1, \ldots, X_m$ *is the vector of all variables in the clause* $\{l_1, \ldots, l_n\}$,
4. $I(\exists\{l_1, \ldots, l_n\})$ [7] $= 1$, *iff there exists a vector* $\boldsymbol{a} = a_1, \ldots, a_m$, $a_i \in \Delta, (1 \leq i \leq m) : I\langle X_1/a_1, \ldots, X_m/a_m\rangle(l_1) = 1$ *or* ... *or* $I\langle X_1/a_1, \ldots, X_m/a_m\rangle(l_n) = 1$, *where* $\boldsymbol{X} = X_1, \ldots, X_m$ *is the vector of all variables in the clause* $\{l_1, \ldots, l_n\}$,
5. $I(\neg A) = 1$ *iff* $I(A) = 0$
6. $I(P_n(t_1, \ldots, t_n)) = 1$ *iff* $t_1^I, \ldots, t_n^I \in P_n^I$
7. $I(C(t)) = 1$ *iff* $t^I \in C^I$, *with* C^I *defined in table 1*
8. $I(R(t_1, t_2)) = 1$ *iff* $t_1^I, t_2^I \in R^I$.

Definition 7 (logical consequence).
A clause C follows from (is a logical consequence of) a clause set P (written $P \models C$), iff all models of P are also models of C. A clause set P_1 follows from (is a logical consequence of) a clause set P_2 (written $P_2 \models P_1$), iff all models of P_2 are also models of P_1.

We also use the normal ILP definition of learning to define learning of description logic programs (called Induction of Description Logic Programs or short IDLP).

Definition 8 (The IDLP Learning Problem).
Given a logical language $\mathcal{L}$ (i.e. our language DLP) with a consequence relation $\models$, background knowledge B in a Language $LB \subseteq \mathcal{L}$, positive and negative examples $E = E^+ \cup E^-$ in a language $LE \subseteq \mathcal{L}$ consistent with B ($B, E \not\models \Box$) and

[5] $I\langle X/a\rangle(l)$ means that during the interpretation $I(l)$ the variable X in the literal l is always interpreted as the domain element a. It must not be confused with a substitution as a domain element a may not have a constant in $\mathcal{F}$ denoting it, i.e. $I^{-1}(a)$ may be undefined.

[6] In normal first-order logic such a unique name assumption for constants is usually not used, but for counting in DLs with number-restrictions it is very useful. It must not be confused with the object identity restriction on variables (substitutions must be injective) also used in some ILP approaches.

[7] This formula is not an expression **in** our language, but we need to interpret it in a lemma **about** our language.

not already a consequence of B ($\forall e \in E : B \not\models e$)*, and a hypothesis language* $LH \subseteq \mathcal{L}$*. Find a hypothesis* $h \in LH$ *such that:*

(I) $(B, h, E \not\models \square)$*, i.e.* h *is consistent with* B *and* E*.*
(II) $(B, h \models E^+)$*, i.e.* h *and* B *explain* E^+*.*
(III) $(B, h \not\models E^-)$*, i.e.* h *and* B *do not explain* E^-*.*

The tuple $(\models, LB, LE, LH)$ *is called the* IDLP learning problem*. Deciding whether there exists such an* $h \in LH$*, is called the* IDLP consistency problem*. An algorithm which accepts any* $B \in LB$ *and* $E \in LE$ *as input and computes such an* $h \in LH$ *if it exists, or "no" if it does not exist is called an* IDLP learning-algorithm.

As we are interested in polynomial learnability results [Cohen, 1995] for DLP, we are especially interested in IDLP problems, where B and E consist of ground (variable-free) facts, and H is restricted to 1-clause Programs (see section 3.4).

3.1 The Relation between (Learning) CARIN-$\mathcal{ALN}$ and (I)DLP

At first this looks quite different from the definition of CARIN-$\mathcal{ALN}$ in [Levy and Rouset, 1998] or the CARIN-$\mathcal{ALN}$ learning problem as defined in [Rouveirol and Ventos, 2000], but it is in fact quite similar. We of course do not claim equivalence of the formalisms, but we think that we have transported the important (to us) ideas from CARIN-$\mathcal{ALN}$ into a more (inductive) logic programming oriented form. The DLP formalism defined above is in general to expressive compared with CARIN-$\mathcal{ALN}$, as so far we have neither forbidden recursion nor DL-literals as the head of rules, two things known to be quite difficult to reason with in CARIN-$\mathcal{ALN}$ [Levy and Rouset, 1998]. But we have to introduce such restrictions as well, as rules like $connected(X, Y) \leftarrow node(X), node(Y), [\leq 0\ connected](X)$ expressible in the formalism are difficult to interpret[8] under the normal model-theoretic semantics we used. But as in ILP we want to start with the general idea, and then introduce the restrictions needed to come to a characterisation of polynomial learnability in the end. Given the goal of learning DLPs, recursive description logic programs are not very interesting anyway, as the set of know polynomial learnable recursive logic programs is very restricted and so far only of interest for synthesis of toy programs and not for real data analysis or mining applications.

The other main difference between DLP and CARIN-$\mathcal{ALN}$ is that we have unified assertional component (the set of facts from our DLP) and rule component (the set of rules from our DLP) of CARIN-$\mathcal{ALN}$, and - as already discussed in section 2 - we have dropped the terminological component of CARIN-$\mathcal{ALN}$, but are instead able to express rules and assertions, where concepts are already expanded with respect to the (acyclic) terminological axioms [Nebel, 1990a], e.g.

[8] They do not have a (stable) model, so they have to be interpreted as contradictory, but an interpretation as default rule may be more adequate. But this is not a topic of this paper.

instead of separating terminological axioms T, rules R and assertions A as in the following example from [Rouveirol and Ventos, 2000]:

$T = \{empty\_car \equiv car \sqcap \leq 0\ has\_load,$
$\quad empty\_train \equiv train \sqcap \forall has\_car.empty\_car\}$
$R = \{east\_train(X) \leftarrow empty\_train(X)\}$
$A = \{train(a),\ has\_car(a,d),\ \leq 1\ has\_car(a),\ empty\_car(d)\}$

we require the representation of the equivalent (with respect to assertional and rule consequences) terminological expanded rule and facts within one description logic program: $train(a).\ has\_car(a,d).\ [\leq 1 has\_car](a).\ [car \sqcap \leq 0\ has\_load](d).$ $east\_train(X) \leftarrow [train \sqcap \forall has\_car.[car \sqcap \leq 0\ has\_load]](X).$

Except for the dropped $\mathcal{ALN}$ terminological component which makes polynomial learnability results possible (with a terminological component learning is at-least as hard as reasoning, i.e coNP-hard [Nebel, 1990b]) this is only a syntactic difference.

Concerning the learning framework in [Rouveirol and Ventos, 2000] and the IDLP problem, the main difference is that they use the learning from interpretation setting, whereas we use the normal ILP setting. For non-recursive clauses, this makes no difference, and they could be easily transformed into each other with only a polynomial size difference, if B and E consist of ground facts, i.e. take the whole B as interpretation for each example, or take the union of all interpretations as B.

3.2 The Relation between (I)LP and (I)DLP

As known from CARIN-$\mathcal{ALN}$, there are also serious differences between DLP and LP. A set of rules may have more consequences than the union of the consequences of the individual rules, e.g. let
$P_1 = \{train(a).\ east\_train(X) \leftarrow [train \sqcap \leq 1\ has\_car](X).\}$ and
$P_2 = \{train(a).\ east\_train(X) \leftarrow [train \sqcap \geq 1\ has\_car](X).\}$.
Neither $P_1 \models east\_train(a)$ nor $P_2 \models east\_train(a)$ but $P_1 \cup P_2 \models east\_train(a)$ as $[\geq 1\ has\_car](X) \vee [\leq 1\ has\_car](X)$ is a tautology.

Theoretically (section 3.4), this does not hinder our transfer of learnability results from LP to DLP very much as most results concerning polynomial learnability of ILP are restricted to one rule programs [Cohen, 1995], but this is a serious problem for applying these positive DLP results practically as done in ILP. Most ILP systems use the greedy strategy of learning rules individually such that no negative examples are covered by any rule until all positive examples are covered by a rule. This difference between DLP and LP means that a rule set may cover negative examples, even if no rule does. However, this is only a problem in learning DLPs from DLP facts, but not in the practically much more important subproblem of learning DLPs from ILP data sets, e.g. from relational databases. Under the usual open world assumption in DLP no rule ever learned from an ILP data set will contain an at-most or all restriction, as under the open-world assumption (OWA) a rule with an at-most or all restriction will never cover an example described by datalog facts only, e.g. P_1 will never cover

a train as without an explicit at-most restriction in the examples, i.e. the OWA means we will never know that we know all cars of a train. This however makes learning DLP quite useless as well, as only at-least restrictions are learnable from ILP facts, but neither at-most nor all restriction.

There is an easy practical way out of this problem used in section 4 to learn DLP programs from ILP data sets namely assuming that the background knowledge contains all relevant background knowledge for the examples (a closed world assumption on B). In that case we can also learn at-most and all restriction and it is always possible to decide which side of a tautology is applicable, e.g. if we know that we know all the cars of the trains under consideration, we can decide by counting, if rule P_1 or P_2 or both are applicable. Also for a primitive concept P we are able to decide, if P or $\neg P$ is true. In summary, we do not have to worry about this reasoning problem with respect to the goals of this paper. Theoretically we only need reasoning with one clause programs, practically we need a closed world assumption to learn DLPs and in both cases this problem disappears.

Another difference between LP and DLP reasoning is that a DLP like the one above also contains implicit knowledge not only due to rule reasoning as usual in logic programing but also due to interactions between facts with description logic literals (DL-literals) and normal literals (HL-literals[9]), i.e. the fact $[\forall has\_car.[car \sqcap \ \leq 0\ has\_load]](a)$ is true in every model of the DLP as a consequence of the interaction between all the above facts and after we have deduced that $east\_train(a)$ can be deduced as a consequence of the facts and the rule.

3.3 Subsumption between DLP Clauses

We have to make such implicit literals explicit, as a reasoning procedure based on substructure matching like subsumption would be incomplete otherwise, e.g. $h(X) \leftarrow a(X,Y) \Leftrightarrow h(X) \leftarrow (\geq 1a)(X)$, but $h(X) \leftarrow a(X,Y) \not\vdash_{\theta_{I\perp}} h(X) \leftarrow \Phi(\geq 1a, X)\ h(X) \leftarrow \Phi(\geq 1a, X) \not\vdash_{\theta_{I\perp}} h(X) \leftarrow a(X,Y)$.

If the rules were ground, the interaction between HL- and DL-terms would correspond to what is called ABox reasoning in description logic, i.e. inferring HL-literals corresponds to ABox completion [Baader and Sattler, 2000] and inferring DL-literals corresponds to computing the most specific concept [Baader and Küsters, 1998] for every variable. Goasdoué, Rouveirol and Ventos [2001] have put them together as completion rules to formalize example saturation for learning under OI-subsumption. The adaptation of their rules to θ-subsumption lead to the following completion rules to be applied to a set of DLP facts, i.e. either background knowledge B or the body B of a DLP rule $H \leftarrow B$.

Definition 9 (Disjointness Clique). *A set of terms $\{X_1, ..., X_n\}$ is called a disjointness clique of size n of a rule body B, iff for all pairs $\{X_i, X_j\} \subseteq$*

[9] Primitive roles and concepts strictly belong to both classes, but we will reference and handle primitive roles and primitive concepts as HL-literals.

$\{X_1, \ldots, X_n\}$, *with* $i \neq j$ *either* X_i *and* X_j *are not unifiable terms, i.e. different constants, or* $\{C_i(X_i), C_j(X_j)\} \subseteq B$ *and* $C_i \sqcap C_j \equiv \bot$, *i.e. there must be no model, where they are interpreted as the same individual.*

Definition 10 (Completion for DLP under θ-Subsumption). *Let* B *a set of DLP facts.*

1. **apply at-least restriction:** *if there exists a substitution* σ *such that* $((\geq n\ r)(X_0))\sigma \in B$ *for* $n \geq 1$ *and* $(\{r(X_0, X_1)\})\sigma \notin B$ *then* $B \rightharpoonup B \cup (\{r(X_0, U)\})\sigma$, *and* U *is a new variable not occurring in* B *(and* σ*) so far.*
2. **apply value restriction:** *if there exists a substitution* σ *such that* $(\{(\forall r.C)(X_0), r(X_0, X_1)\})\sigma \subseteq B$ *and* $C(X_1)\sigma \notin B$ *then* $B \rightharpoonup B \cup \{C(X_1)\sigma\}$.
3. **infer at-least restriction:** *If there exists a substitution* σ *such that*
   - $(\{r(X_0, X_1), \ldots, r(X_0, X_n)\})\sigma \subseteq B$, *and there is no further* $r(X_0, X_{n+1})\sigma$ *in* B, *and*
   - k *is size of the maximal disjointness clique of* $\{X_1, \ldots, X_n\}\sigma$ *in* B, *and*
   - $((\geq m\ r)(X_0))\sigma \notin B$ *for any* $m \geq k$

   then $B \rightharpoonup B \cup \{[\geq k\ r](X_0)\}\sigma$.
4. **infer value restriction:** *If there exists a substitution* σ *such that* $(\{(\leq n\ r)(X_0), r(X_0, X_1), \ldots, r(X_0, X_n), C_1(X_1), \ldots, C_n(X_n)\})\sigma \subseteq B$, *and* $\{X_1, ..., X_n\}\sigma$ *is a disjointness clique is of size* n, *let* $C = lcs(C_1, \ldots, C_n)$ *and* $((\forall r.C)(X_0))\sigma \notin B$ *then* $B \rightharpoonup B \cup \{[\forall r.C](X_0)\}\sigma$,
5. **collect and normalize concept-terms over the same variable:** *if* $C(X_0)\sigma \in B$ *and* $D(X_0)\sigma \in B$ *with* C *and* D *are DL-terms then* $B \rightharpoonup B \setminus \{C(X_0)\sigma, D(X_0)\sigma\} \cup \{[norm(C \sqcap D)](X_0)\}\sigma$

This corresponds to ABox reasoning presented as equivalent to tableux-based terminological component reasoning in [Baader and Sattler, 2000], but for learning a tableux reasoning approach which does only consistency checking is not sufficient. We need a constructive normalisation approach to have the consequences available as input for learning. This of course is only feasible for a simple DL like $\mathcal{ALN}$. It in fact corresponds to inferring for each term t occurring in B a depth-bounded acyclic approximation of it's (cyclic) most specific concept as done for $\mathcal{ALN}$-ABoxes in [Baader and Küsters, 1998].

Lemma 2 (The completion rules are correct). *For any* B, *if* $B \rightharpoonup B'$, *then* $\exists B \models \exists B'$. *Proof in [Kietz, 2002]*

Lemma 3 (The completion rules are complete). *For all atoms* $a \in AT$, *whenever* B_0 *is consistent* ($\exists B_0 \not\models \exists \bot(X)$), *if* $\exists B_0 \models \exists a$, *there exits a chain of* $B_0 \rightharpoonup B_1 \rightharpoonup \ldots \rightharpoonup B_n$ *such that for some substitution* σ *either* $a\sigma \in B_n\sigma$ *or* $a\sigma = C_1(t)$ *and there exist a* $C_2(t)\sigma \in B_n\sigma$ *and* $C_2 \sqsubseteq C_1$. *Proof in [Kietz, 2002]*

Theorem 2 (Subsumption between completed encoded rules is sound). *Let* $depth(C)$ *the depth of the deepest* $\forall$ *nesting in* C, *let* $\varphi(D, i)$ *denote the completion of* D *up to depth* i, *i.e. the application of the rules 1-5 as often as possible, without generating a* $\forall$ *nesting deeper than* i *and* Φ_r *the extension of* Φ *to rules, such that all DL-literals* $DL(X)$ *in the rule are encoded with*

$\Phi(norm(DL), X)$ and everything else is returned unchanged. A non-recursive DLP rule C implies a non-tautological DLP rule D ($C \models D$), if and only if the encoding Φ_r of C $\theta_{I\perp}$-subsumes the encoding Φ_r of the completion φ of D ($\Phi_r(C) \vdash_{\theta_{I\perp}} \Phi_r(\varphi(D, depth(C)))$). Proof in [Kietz, 2002]

Theorem 3 (Completion is polynomial for depth-bounded DL terms). *Proof in [Kietz, 2002]*

As Φ and φ are polynomial for ground clauses, the complexity of $C \models D$ is that of θ-subsumption, if C is ground. From [Kietz and Lübbe, 1994] we know, that θ-subsumption is NP-complete in general and polynomial for an arbitrary horn clause D, if $C = C_0 \leftarrow C_{DET}, LOC_1, \ldots, LOC_n$ is a horn clause where $C_0 \leftarrow C_{DET}$ is determinate[10] with respect to D and each $LOC_i, 1 \leq i \leq n$, is a k–local[11]. In that case, $C \vdash_\theta D$ can be decided with $O(\|C\| * \|C_{DET}\| * \|D\| + \|LOC_1, \ldots, LOC_n\|^2 + n * (k^k * \|D\|))$ unification attempts.

Theorem 4 (Simulation of lgg). *Let $E = \Phi_r^{-1} lgg_{I\perp}(\Phi_r(\varphi(C, i)), \Phi_r(\varphi(D, i)))$. E is the least general generalization (lgg) with depth at-most i of C and D, i.e. $E \models C$ and $E \models D$ and if there is an E' with depth at most i with $E' \models C$ and $E' \models D$, then $E' \models E$* [12].

3.4 Learning CARIN-$\mathcal{ALN}$

With the definition of learning and theorem 2 we can immediately conclude on the learnability of DLP programs consisting of a single rule.

Corollary 1. *A IDLP rule learning problem ($\models$,ground DLP facts, ground HL facts, $1-$clause DLP programms) is polynomially learnable, if and only if the corresponding ILP learning problem ($\models$, $\Phi_r(\varphi$(ground DLP facts,i)), ground HL facts, $\Phi_r(\varphi(1-$clause DLP programms,i))) is polynomial learnable.*

As the boarder line of polynomial learnability is quite well known for ILP due to a lot of positive and negative learnability results (see [Kietz, 1996; Kietz and Džeroski, 1994; Cohen and Page, 1995] for overviews), we are now able to characterize it for DLP rules as well. One of the most expressive (single horn clause) ILP-problem that is polynomial learnable, is learning a i, j-determinate-k-literal-local horn clause (see [Kietz and Lübbe, 1994] for definitions and discussions of these restrictions).

Theorem 5. *A 1-clause DLP program, where the HL part is i, j-determinate-k-literal-local and DL-terms are depth-bounded by i and are only allowed on head or determinate variables is polynomial learnable*[13].

[10] The n-ary generalisation of what is called an attribute in DL, i.e. there is a determinate way to find a unique match of the output variables.

[11] It does not share variables with another local, which not also occur in C_0 or C_{DET} and it has at-most k literals (k–literal local) or at-most k variables (k–variable local).

[12] Without depth limit the lgg may not exist, i.e. is infinite due to rule 4.

[13] This includes that they are PAC learnable and polynomial predictable, as Φ is a polynomial prediction preserving reduction.

Proof. The encoding of such a clause is a $2 * i, j$-determinate-k-literal-local horn clause as the DL-term encodings are in itself determinate, start on a determinate variables and are bound in depth by i as well and as such they are polynomial learnable [Cohen, 1995]. □

4 Applying the Results

This encoding does not only produce theoretical results it also works in practise. Let us demonstrate this with the ILP dataset (available from MLNet) for MESH-Design [Džeroski and Dolsak, 1992].

4.1 Learning from Databases Using Closed World Assumption

As discussed in section 3.2, datalog facts or databases are not adequate to learn DLPs under OWA and the CWA is quite natural a way out for learning [Helft, 1989] as well as for databases. We do not want to close the world totally, but just locally, i.e we assume that if an object is described at all it is described completely, but we don't want to assume that we know about all objects of any world. The following two non-monotonic rules are doing just that.

6. **infer at-most restriction:** If there exists a substitution σ, such that
   - $(\{r(X_0, X_1), \ldots, r(X_0, X_n)\})\sigma \subseteq B$, and there is no further $r(X_0, X_{n+1})\sigma$ in B, and
   - k is size of the maximal disjointness clique of $\{X_1, \ldots, X_n\}\sigma$ in B, and
   - there is no $[\leq m\ r](X_0)\sigma$ in B such that $m \leq k$

   then $B := B \cup \{[\leq m\ r](X_0)\}\sigma$.
7. **infer concept negation:** Let c be a constant appearing in some literal of B, and P be any primitive concept, if $P(c) \notin B$, then $B := B \cup \{\neg P(c)\}$.

Given N constants (or variables) in the rule, and M primitive concepts, only $N * M$ additional literals are introduced at the literal level by the local CWA rule 7, i.e. only a polynomial amount. An $\forall r.\neg P$ can only be added, by rule 4 out of the introduced literals. But $\forall r.\neg P$ does not follow for every role under CWA, i.e under CWA only the literal $\neg P(b)$ is added by rule 7 to $R(a, b), R(a, c), P(c)$, but $(\forall r.\neg P)(a)$ is obviously not true and not added. Theorem 2 proves polynomial complexity in the size of the input of rule 4 under OWA. Therefore the complexity under CWA is polynomial as well.

4.2 Learning MESH-Design in DLP

We have chosen MESH-Design as it only contains unary and binary predicates, i.e. perfectly fits description logic without the need for further preprocessing. We have chosen CILGG [Kietz, 1996] as the ILP-systems to learn the encoding descriptions as it has interval handling, is optimized for determinate literals (e.g. like Golem [Muggleton and Feng, 1992]) and is able to learn k-literal-local clauses completely for small k. As depth limit we had chosen 1 for HL-literals. In MESH-Design this produces 4-literal-local ground starting (bottom) clauses. We

```
mesh(A,2):- [usual ⊓
             ≥ 2 neighbour ⊓
             ≤ 3 neighbour ⊓
             ∀ neighbour.not_loaded
             ](A).
```

```
mesh(A,2):- [usual ⊓
             ≥ 1 opposite ⊓
             ≤ 1 opposite ⊓
             ∀ opposite.[fixed ⊓
                         ≥ 2 opposite ⊓
                         ≤ 2 opposite
                         ]
             ](A).
```

Fig. 1. Two of the description logic rules learned for MESH-Design

Table 2. Comparison of ILP-Approaches learning the MESH-Data set

	A	B	C	D	E	Σ	%	Avg.
Maximum	52	38	28	57	89	268		CPU
Default	9	9	6	13	23	60	22	Time
Foil	17	5	7	9	5	43	16	(5m, in 1992)
mFoil	22	12	9	6	10	59	22	(2h, in 1992)
Golem	17	9	5	11	10	52	20	(1h, in 1992)
Indigo	21	14	9	18	33	95	36	(9h, in 1996)
Claudien	31	9	5	19	15	79	30	(16m, in 1992)
CILGG(1996)	19	16	6	10	9	60	22	(85s, in 1996)
CILGG DL	16	8	5	10	12	51	19	8s
CILGG HL	22	14	8	13	5	62	23	11s
CILGG CL	19	16	7	14	23	79	30	22s
CILGG 20-NN DL	20	12	9	16	38	95	36	19s
CILGG 20-NN HL	17	14	9	18	41	99	38	23s
CILGG 20-NN CL	26	12	10	18	37	103	39	52s

restricted DL-terms to the head variable/object as this is the only determinate one and to the depth of 3, as deeper terms are very difficult to understand. We have made three experiments, one with only the literals generated from the DL-term, one with only the HL-literals, and one with both (CL). CILGG [Kietz, 1996] has two possibilities to use the learned rules for testing, the normal deductive one for most general discriminations (generated similar as in Golem from the learned lggs), and a k-nearest neighbor classification (20-NN in this case) using the learned lggs. The results are in table 2 together with the results reported in [Džeroski and Dolsak, 1992] and Indigo [personal note from P. Geibel, 1996]. The table gives the number of correctly (and uniquely) classified edges per object using rules learned from the other objects. The results indicates that the extended language helps to learn better rules in MESH-Design. The used runtime also reflects the theoretical result that learning the DL-part is easier than learning the HL-part. However, this single experiment is not sufficient to claim the usefulness of CARIN-$\mathcal{ALN}$ as a hypothesis language in general. But it clearly shows, that this gain in expressivity of the language can help to achive better results, and that the price to pay for this gain in terms of complexity and computational costs is not very high, i.e. it enables normal ILP-system to learn rules like the ones in Figure 1 efficiently, which are not in the normal ILP bias and which may be useful in some application domains.

5 Summary and Outlook

We have characterizes the boarder line of polynomial learnability of CARIN-$\mathcal{ALN}$ rules from ground facts by reducing it to the well investigated boarder-line of polynomial learnability in ILP. This work should be extended to more expressive forms of background knowledge, e.g. terminological axioms (an ontology) and description logic programs. We also showed in a first experiment, that this theoretical encoding is applicable in practice. However, careful experimentation about the usefulness of using CARIN-$\mathcal{ALN}$ as hypothesis language is still missing. But, we provided a data preprocessing method, that allows us to do that with a broad range of ILP-systems on a broad range of ILP-applications. On the theoretical side the IDLP framework could serve to analyse the learnability of the related new aggregation-based ILP approaches [Krogel and Wrobel, 2001] .

Acknowledgements

This work has been partially founded by the Swiss Government (BBW Nr.99.0158) as part of the European commission Research Project Mining Mart (IST-1999-11993) while the author worked at Swiss Life. The presentation of the paper was founded by the European commission NoE-project ILPNet2 (INCO 977102). I thank François Goasdoué, Ralf Molitor, Céline Rouveirol and Véronique Ventos for very helpful and interesting discussions about this topic and earlier drafts of this paper.

References

[Baader and Küsters, 1998] Baader, F. and R. Küsters: 1998, 'Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$-concept descriptions'. In: O. Herzog and A. Günter (eds.): *Proceedings of the 22nd Annual German Conference on Artificial Intelligence, KI-98*. pp. 129–140, Springer–Verlag.

[Baader and Sattler, 2000] Baader, F. and U. Sattler: 2000, 'Tableau Algorithms for Description Logics'. In: R. Dyckhoff (ed.): *Proceedings of the International Conference on Automated Reasoning with Tableaux and Related Methods (Tableaux 2000)*. pp. 1–18, Springer-Verlag.

[Borgida, 1996] Borgida, A.: 1996, 'On the relative expressiveness of description logics and predicate logics'. *Artificial Intelligence* **82**, 353 – 367.

[Brachman and Schmolze, 1985] Brachman, R. J. and J. G. Schmolze: 1985, 'An Overview of the KL-ONE Knowledge Representation System'. *Cognitive Science* **9**(2), 171 – 216.

[Cohen and Page, 1995] Cohen, W. and C. Page: 1995, 'Polynomial Learnability and Inductive Logic Programming: Methods and Results'. *New Generation Computing, Special issue on Inductive Logic Programming* **13**(3-4), 369–410.

[Cohen, 1995] Cohen, W. W.: 1995, 'Pac-Learning non-recursive Prolog Clauses'. *Artificial Intelligence* **79**, 1–38.

[Cohen et al., 1992] Cohen, W. W., A. Borgida, and H. Hirsh: 1992, 'Computing Least Common Subsumers in Description Logic'. In: *Proc. of the 10th National Conference on Artificial Intelligence*. San Jose, California, MIT–Press.

[Cohen and Hirsh, 1994] Cohen, W. W. and H. Hirsh: 1994, 'The Learnability of Description Logics with Equality Constraints'. *Machine Learning* **17**, 169–199.

[Donini et al., 1991] Donini, F., M. Lenzerini, C. Nardi, and W. Nutt: 1991, 'Tractable Concept Languages'. In: *Proc. IJCAI-91.* pp. 458–463.

[Džeroski and Dolsak, 1992] Džeroski, S. and B. Dolsak: 1992, 'A Comparision of Relation Learning Algorithms on the Problem of Finite Element Mesh Design'. In: *Proc. of the ISEEK Workshop.* Ljubljana, Slovenia.

[Frazier and Pitt, 1994] Frazier, M. and L. Pitt: 1994, 'Classic Learning'. In: *Proc. of the 7th Annual ACM Conference on Computational Learning Theory.* pp. 23–34.

[Goasdoué et al., 2001] Goasdoué, F., C. Rouveirol, and V. Ventos: 2001, 'Optimized Coverage Test for Learning in CARIN-$\mathcal{ALN}$'. Technical report, L.R.I, C.N.R.S and Université Paris Sud. Work in progress.

[Helft, 1989] Helft, N.: 1989, 'Induction as nonmonotonic inference'. In: *Proceedings of the 1st International Conference on Knowledge Representation and Reasoning.*

[Kietz, 1996] Kietz, J.-U.: 1996, 'Induktive Analyse Relationaler Daten'. Ph.D. thesis, Technical University Berlin. (in german).

[Kietz, 2002] Kietz, J.-U.: 2002, 'Learnability of Description Logic Programs (Extended Version)'. Technical report, http://www.kietz.ch/.

[Kietz and Džeroski, 1994] Kietz, J.-U. and S. Džeroski: 1994, 'Inductive Logic Programming and Learnability'. *SIGART Bulletin* **5**(1).

[Kietz and Lübbe, 1994] Kietz, J.-U. and M. Lübbe: 1994, 'An Efficient Subsumption Algorithm for Inductive Logic Programming'. In: *Proc. of the Eleventh International Conference on Machine Learning (ML94).*

[Kietz and Morik, 1994] Kietz, J.-U. and K. Morik: 1994, 'A polynomial approach to the constructive Induction of Structural Knowledge'. *Machine Learning* **14**(2), 193–217.

[Krogel and Wrobel, 2001] Krogel, M. A. and S. Wrobel: 2001, 'Transformation-based Learning Using Mulirelational Aggregation'. In: *Proc. Elenth International Conference on Inductive Logic Programming, ILP'2001.* Berlin, New York, Springer Verlag.

[Levy and Rouset, 1998] Levy, A. Y. and M.-C. Rouset: 1998, 'Combining horn rules and description logic in CARIN'. *Artificial Intelligence* **104**, 165–209.

[Muggleton, 1995] Muggleton, S. H.: 1995, 'Inverse Entailment and Progol'. *New Generation Computing* **13**.

[Muggleton and Feng, 1992] Muggleton, S. H. and C. Feng: 1992, 'Efficient induction of logic programs'. In: S. H. Muggleton (ed.): *Inductive Logic Programming.* Academic Press.

[Nebel, 1990a] Nebel, B.: 1990a, *Reasoning and Revision in Hybrid Representation Systems.* New York: Springer.

[Nebel, 1990b] Nebel, B.: 1990b, 'Terminological reasoning is inherently intractable'. *Artificial Intelligence* **43**, 235 – 249.

[Plotkin, 1970] Plotkin, G. D.: 1970, 'A note on inductive generalization'. In: B. Meltzer and D. Michie (eds.): *Machine Intelligence*, Vol. 5. American Elsevier, Chapt. 8, pp. 153 – 163.

[Quinlan and Cameron-Jones, 1993] Quinlan, R. and R. M. Cameron-Jones: 1993, 'FOIL: A Midterm Report'. In: P. Brazdil (ed.): *Proceedings of the Sixth European Conference on Machine Leaning (ECML-93).* Berlin, Heidelberg, pp. 3–20, Springer Verlag.

[Rouveirol and Ventos, 2000] Rouveirol, C. and V. Ventos: 2000, 'Towards learning in CARIN-$\mathcal{ALN}$'. In: J. Cussens and A. M. Frisch (eds.): *Proc. Tenth International Conference on Inductive Logic Programming, ILP'2000.* Berlin, Springer Verlag.

[Sebag and Rouveirol, 1996] Sebag, M. and C. Rouveirol: 1996, 'Constraint Inductive Logic Programming'. In: L. de Raedt (ed.): *Advances in ILP.* IOS Press.

1BC2: A True First-Order Bayesian Classifier

Nicolas Lachiche[1] and Peter A. Flach[2]

[1] LSIIT - Université Robert Schuman, France
lachiche@lsiit.u-strasbg.fr
[2] Department of Computer Science, University of Bristol, United Kingdom
Peter.Flach@bristol.ac.uk

Abstract. In previous work [3] we presented 1BC, a first-order Bayesian classifier. 1BC applies dynamic propositionalisation, in the sense that attributes representing first-order features are generated exhaustively within a given feature bias, but during learning rather than as a pre-processing step. In this paper we describe 1BC2, which learns from structured data by fitting various parametric distributions over sets and lists to the data. We evaluate the feasibility of the approach by various experiments.

1 Introduction

In its general form, a Bayesian classifier predicts the most likely class value c of an object given its description d: $argmax_c P(c|d)$. Applying Bayes theorem, this formula can be turned into: $argmax_c \frac{P(d|c)P(c)}{P(d)} = argmax_c P(d|c)P(c)$. In typical machine learning problems, the number of possible descriptions of individuals is much greater than the number of available examples, therefore it is impossible, in general, to get a good estimate of $P(d|c)$. This is the usual bias-variance tradeoff [4]: a fundamental challenge of constructing classifiers is finding the right complexity of hypothesis (greater complexity generally decreases bias but increases variance) for a given amount of data (more data decreases variance). In this paper, we will consider a single change of complexity, relying on the naive Bayes assumption.

In an attribute-value representation, the object is described by its values $a_1, \ldots, a_n$ for a fixed set of attributes $A_1, \ldots, A_n$. Since it is difficult to get a good estimate of $P(a_1, \ldots, a_n|c)$, the naive Bayes assumption consists in decomposing this probability distribution into the product of the probability of each attribute value: $P(a_1|c) \times \ldots \times P(a_n|c)$, assuming that the probability of attribute A_i taking on value a_i is independent of the probabilities of the values taken by the other attributes. Roughly, since the counting of the examples of class c having the description $a_1, \ldots, a_n$ is difficult, it is evaluated by counting the examples of class c having the description a_i and by combining those estimates.

While the decomposition of probability distributions on attribute-value tuples is well understood, we are not aware of any studies of the decomposition of probability distributions on structured individuals. There have been works on Bayesian classification on structured individuals. Pompe and Kononenko describe an application of naive Bayesian classifiers in a first-order context [9]. Their approach consists in learning a set of first-order rules in a first step, then using them as new attributes in a classical attribute-value

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 133–148, 2003.

naive Bayesian classifier. The 1BC system we developed [3] does not learn first-order rules, it generates instead a set of first-order conditions, that are used then as attributes in a classical attribute-value naive Bayesian classifier. It can thus be classified as a propositionalisation approach (although the propositionalisation is done dynamically, not in a pre-processing step as in LINUS [7], and it considers only satisfied features, i.e. the one occuring in the individual). In this paper we propose a different approach that is less 'propositional' and directly considers probability distributions on structured individuals made of sets, tuples and multisets.

We start by considering probability distributions over first- and higher-order terms in Section 2. In particular, we obtain a distribution over sets which is different from the standard bitvector distribution in that it only depends on the probabilities of its elements. Section 3 describes the 1BC2 system including its main algorithm and the declarative bias it employs. Section 4 describes the results of our experiments. Section 5 concludes.

2 Probability Distributions over Lists and Sets

In this section we assume a finite *alphabet* $A = \{x_1, \ldots, x_n\}$ of atomic objects (e.g. integers or characters – we are only dealing with categorical data analysis in this paper), and we consider the question: how to define probability distributions ranging over lists and sets of elements from A?

2.1 Probability Distributions over Lists

We can define a uniform probability distribution over lists if we consider only finitely many of them, say, up to and including length L. There are $\frac{n^{L+1}-1}{n-1}$ of those for $n > 1$, so under a uniform distribution every list has probability $\frac{n-1}{n^{L+1}-1}$ for $n > 1$, and probability $\frac{1}{L+1}$ for $n = 1$. Clearly, such a distribution does not depend on the internal structure of the lists, treating each of them as equiprobable.

A slightly more interesting case includes a probability distribution over lengths of lists. This has the additional advantage that we can define distributions over all (infinitely many) lists over A. For instance, we can use the geometric distribution over list lengths: $P_\tau(l) = \tau(1-\tau)^l$, with parameter τ denoting the probability of the empty list. Of course, we can use other infinite distributions, or arbitrary finite distributions, as long as they sum up to 1. The geometric distribution corresponds to the head-tail representation of lists.

We then need, for each list length l, a probability distribution over lists of length l. We can again assume a uniform distribution: since there are n^l lists of length l, we would assign probability n^{-l} to each of them. Combining the two distributions over list lengths and over lists of fixed length, we assign probability $\tau(\frac{1-\tau}{n})^l$ to any list of length l. Such a distribution only depends on the length of the list, not on the elements it contains.

We can also assume a probability distribution P_A over the alphabet, and use this to define a non-uniform distribution over lists of length l. For instance, among the lists of length 3, list $[a, b, c]$ would have probability $P_A(a)P_A(b)P_A(c)$, and so would its 5 permutations. Combining P_A and P_τ thus gives us a distribution over lists which depends on the length and the elements of the list, but ignores their positions or ordering.

Definition 1 (Distribution over lists). *The following defines a probability distribution over lists:*

$$P_{\text{li}}([x_{j_1},\ldots,x_{j_l}]) = \tau(1-\tau)^l \prod_{i=1}^{l} P_A(x_{j_i})$$

where $0 < \tau \leq 1$ *is a parameter determining the probability of the empty list.*

Introducing an extended alphabet $A' = \{\epsilon, x_1, \ldots, x_n\}$ and a renormalised distribution $P_{A'}(\epsilon) = \tau$ and $P_{A'}(x_i) = (1-\tau)P_A(x_i)$, we have $P_{\text{li}}([x_{j_1},\ldots,x_{j_l}]) = P_{A'}(\epsilon)\prod_{i=1}^{l} P_{A'}(x_{j_i})$. That is, under P_{li} we can view each list as an infinite tuple of finitely many independently chosen elements of the alphabet, followed by the stop symbol ϵ representing an infinite empty tail.

Example 1 (Order-independent distribution over lists). Consider an alphabet $A = \{a, b, c\}$, and suppose that the probability of each element occurring is estimated as $P_A(a) = .2$, $P_A(b) = .3$, and $P_A(c) = .5$. Taking $\tau = (1-.2)(1-.3)(1-.5) = .28$, i.e. using the bitvector estimate of an empty list, but another estimate could be used (cf. 3.2), we have $P_{A'}(a) = (1-.28)*.2 = .14$, $P_{A'}(b) = .22$, and $P_{A'}(c) = .36$, and $P_{\text{li}}([a]) = .28*.14 = .04$, $P_{\text{li}}([b]) = .06$, $P_{\text{li}}([c]) = .10$, $P_{\text{li}}([a,b]) = .28*.14*.22 = .009$, $P_{\text{li}}([a,c]) = .014$, $P_{\text{li}}([b,c]) = .022$, and $P_{\text{li}}([a,b,c]) = .28*.14*.22*.36 = .003$. We also have, e.g., $P_{\text{li}}([a,b,b,c]) = .28*.14*.22*.22*.36 = .0007$.

If we want to include the ordering of the list elements in the distribution, we can assume a distribution P_{A^2} over pairs of elements of the alphabet, so that $[a,b,c]$ would have probability $P_{A^2}(ab)P_{A^2}(bc)$ among the lists of length 3. To include lists of length 0 and 1, we can add special start and stop symbols to the alphabet. Such a distribution would take some aspects of the ordering into account, but note that $[a,a,b,a]$ and $[a,b,a,a]$ would still obtain the same probability, because they consist of the same pairs (aa, ab, and ba), and they start and end with a. Obviously we can continue this process with triples, quadruples etc., but note that this is both increasingly computationally expensive and unreliable if the probabilities must be estimated from data.

A different approach is obtained by taking not ordering but position into account. For instance, we can have three distributions $P_{A,1}$, $P_{A,2}$ and $P_{A,3+}$ over the alphabet, for positions 1, 2, and 3 and higher, respectively. Among the lists of length 4, the list $[a,b,c,d]$ would get probability $P_{A,1}(a)P_{A,2}(b)P_{A,3+}(c)P_{A,3+}(d)$; so would the list $[a,b,d,c]$.

In summary, all except the most trivial probability distributions over lists involve (i) a distribution over lengths, and (ii) distributions over lists of fixed length. The latter take the list elements, ordering and/or position into account. We do not claim any originality with respect to the above distributions, as these and similar distributions are commonplace in e.g. bioinformatics. In the next section we use list distributions to define distributions over sets and multisets. Notice that, while lists are first-order terms, sets represent predicates and therefore are higher-order terms. In this respect our work extends related work on probability distributions over first-order terms.

2.2 Probability Distributions over Sets and Multisets

A multiset (also called a bag) differs from a list in that its elements are unordered, but multiple elements may occur. Assuming some arbitrary total ordering on the alphabet, each multiset has a unique representation such as $\{[a,b,b,c]\}$. Each multiset can be mapped to the equivalence class of lists consisting of all permutations of its elements. For instance, the previous multiset corresponds to 12 lists. Now, given any probability distribution over lists that assigns equal probability to all permutations of a given list, this provides us with a method to turn such a distribution into a distribution over multisets. In particular, we can employ P_{li} above which defines the probability of a list, among all lists with the same length, as the product of the probabilities of their elements.

Definition 2 (Distribution over multisets). *For any multiset s, let l stand for its cardinality, and let k_i stand for the number of occurrences of the i-th element of the alphabet. The following defines a probability distribution over multisets:*

$$P_{\mathrm{ms}}(s) = \frac{l!}{k_1!\ldots k_n!} P_{\mathrm{li}}(s) = l!\tau \prod_i \frac{P_{A'}(x_i)^{k_i}}{k_i!}$$

where τ is a parameter giving the probability of the empty multiset.

Here, $\frac{l!}{k_1!\ldots k_n!}$ stands for the number of permutations of a list with possible duplicates.

Example 2 (Distribution over multisets). Continuing Example 1, we have $P_{\mathrm{ms}}(\{[a]\}) = .04$, $P_{\mathrm{ms}}(\{[b]\}) = .06$, $P_{\mathrm{ms}}(\{[c]\}) = .10$ as before. However, $P_{\mathrm{ms}}(\{[a,b]\}) = .02$, $P_{\mathrm{ms}}(\{[a,c]\}) = .03$, $P_{\mathrm{ms}}(\{[b,c]\}) = .04$, $P_{\mathrm{ms}}(\{[a,b,c]\}) = .02$, and $P_{\mathrm{ms}}(\{[a,b,b,c]\}) = .008$.

This method, of defining a probability distribution over a type by virtue of that type being isomorphic to a partition of another type for which a probability distribution is already defined, is more generally applicable. Although in the above case we assumed that the distribution over each block in the partition is uniform, so that we only have to count its number of elements, this is not a necessary condition. Indeed, blocks in the partition can be infinite, as long as we can derive an expression for its cumulative probability. We will now proceed to derive a probability distribution over sets from distribution P_{li} over lists in this manner.

Consider the set $\{a,b\}$. It can be interpreted to stand for all lists of length at least 2 which contain (i) at least a and b, and (ii) no other element of the alphabet besides a and b. The cumulative probability of lists of the second type is easily calculated.

Lemma 1. *Consider a subset S of l elements from the alphabet, with cumulative probability $P_{A'}(S) = \sum_{x_i \in S} P_{A'}(x_i)$. The cumulative probability of all lists of length at least l containing only elements from S is $f(S) = \tau \frac{(P_{A'}(S))^l}{1-P_{A'}(S)}$.*

Proof. We can delete all elements in S from the alphabet and replace them by a single element x_S with probability $P_{A'}(S)$. The lists we want consist of l or more occurrences of x_S. Their cumulative probability is

$$\sum_{j \geq l} \tau (P_{A'}(S))^j = \tau \frac{(P_{A'}(S))^l}{1-P_{A'}(S)}$$

$f(S)$ as defined in Lemma 1 is not a probability, because the construction in the proof includes lists that do not contain all elements of S. For instance, if $S = \{a, b\}$ they include lists containing only a's or only b's. More generally, for arbitrary S the construction includes lists over every possible subset of S, which have to be excluded in the calculation of the probability of S. In other words, the calculation of the probability of a set iterates over its subsets.

Definition 3 (Subset-distribution over sets). *Let S be a non-empty subset of l elements from the alphabet, and define*

$$P_{\text{ss}}(S) = \sum_{S' \subseteq S} (-P_{A'}(S'))^{l-l'} * f(S')$$

where l' is the cardinality of S', and $f(S')$ is as defined in Lemma 1. Furthermore, define $P_{\text{ss}}(\emptyset) = \tau$. $P_{\text{ss}}(S)$ is a probability distribution over sets.

Example 3 (Subset-distribution over sets). Continuing Example 2, we have $P_{\text{ss}}(\emptyset) = \tau = .28$, $P_{\text{ss}}(\{a\}) = f(\{a\}) = \tau \frac{P_{A'}(\{a\})}{1-P_{A'}(\{a\})} = .05$, $P_{\text{ss}}(\{b\}) = .08$, and $P_{\text{ss}}(\{c\}) = .16$. Furthermore, $P_{\text{ss}}(\{a,b\}) = f(\{a,b\}) - P_{A'}(\{a\}) * f(\{a\}) - P_{A'}(\{b\}) * f(\{b\}) = .03$, $P_{\text{ss}}(\{a,c\}) = .08$, and $P_{\text{ss}}(\{b,c\}) = .15$. Finally, $P_{\text{ss}}(\{a,b,c\}) = f(\{a,b,c\}) - P_{A'}(\{a,b\}) * f(\{a,b\}) - P_{A'}(\{a,c\}) * f(\{a,c\}) - P_{A'}(\{b,c\}) * f(\{b,c\}) + (P_{A'}(\{a\}))^2 * f(\{a\}) + (P_{A'}(\{b\}))^2 * f(\{b\}) + (P_{A'}(\{c\}))^2 * f(\{c\}) = .18$.

P_{ss} takes only the elements occurring in a set into account, and ignores the remaining elements of the alphabet. For instance, the set $\{a, b, c\}$ will have the same probability regardless whether there is one more element d in the alphabet with probability p, or 10 more elements with cumulative probability p. This situation is analogous to lists.

In this section, we defined probability distributions over lists, sets and multisets. Those probability distributions allowed us to build up a true first-order naive Bayesian classifier that can deal with structured data involving nested tuples, lists, sets and multisets: `1BC2`.

3 The `1BC2` System

In this section we describe the main features of the `1BC2` system: first the flattened Prolog representation and declarations used to describe the input data, then the main algorithm to fit parametrised probability distributions to hierarchically structured training data. A typewriter font will be used to denote examples of data and algorithms.

3.1 Data Representation

`1BC2` makes use of a data representation similar to the one of `1BC` [3], i.e. it distinguishes structural predicates from properties.

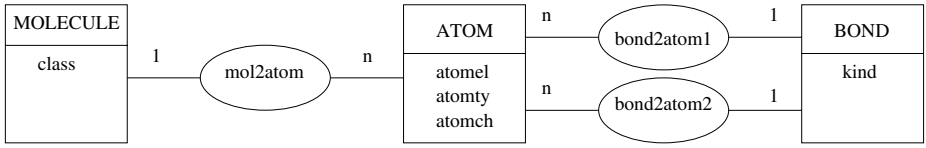


Fig. 1. ER representation of mutagenesis.

Definition 4 (Structural predicate). *A* structural predicate *is a binary predicate representing the relation between one type and another. Given an individual of one type, a* functional *structural predicate, or* structural function*, refers to a unique individual of the other type, while a* non-determinate *structural predicate is non-functional.*

Definition 5 (Property). *A* property *is a predicate characterising an individual. A* parameter *is an argument of a property which is always instantiated. If a property has no parameter (or only one instantiation of its parameters), it is* boolean*, otherwise it is* multivalued*.*

`1BC2` makes use of an ISP (Individual-Structural-Properties) declaration where all structural predicates and properties are declared. Actually the declaration of the individuals is optional since they can be infer from the other declarations: Indeed individuals are non parameter arguments of properties and arguments of structural predicates. Here is the ISP declaration for mutagenesis:

```
--INDIVIDUAL
mol 1 mol
--STRUCTURAL
mol2atom 2 1:mol *:atom 1 li
bond2atom1 2 *:bond 1:atom 1 li
bond2atom2 2 *:bond 1:atom 1 li
--PROPERTIES
class 2 mol #class 1
atomel 2 atom #element 1
atomty 2 atom #number 1
atomch 2 atom #charge 1
kind 2 bond #kind 1
```

Parameters are denoted by a hash: `class, element, number, charge`, and `kind`. Other domains (`mol, atom`, and `bond`) are considered as individuals. Each line defining one predicate contains: the predicate name, its arity, the domain of each argument, and the number of times this predicate can be used (useful either to disable some predicates, or to limit loops with structural predicates). Structural predicates requires one more column: the kind of decomposition that should be used (bv, li, ms, ss). Non-determinate structural predicates are denoted with a star when several individuals (e.g. atoms) can be linked to a single individual (e.g. molecule).

This data representation is in between an Entity-Relationship model (cf. Figure 1) and a relational model [6]: each entity is represented by one individual, it can have some

properties, and it can have some relationships (structural predicates) with other entities. For instance, in mutagenesis domain, molecules involve atoms and bonds. So there are three individuals: `mol, atom` and `bond`. To keep a representation compatible with previous representations, in order to compare our results, we consider three structural predicates: `mol2atom`, `bond2atom1`, and `bond2atom2`. The first structural predicate links a molecule to its atoms, the second and third structural predicates respectively link a bond to its first atom and to its second atom. Indeed in previous representations, bonds were represented by an ordered pair of atoms. Finally, the molecule has a single property: its `class`. An atom has three separate properties: its atomic element `atomel`, its atomic type `atomy` and its `charge`. A bond is characterised by a single property: its `kind`.

3.2 The 1BC2 Algorithm

1BC2 is given a target predicate, e.g. class. Let us first detail the classification phase, then the learning phase.

The main function is `getClassDistribution(individual i)`. It returns a table with an estimate of the probabilities of individual `i` for each class. We will use uppercase P to denote probabilities (actually they are the Laplace estimates from counts).

```
getClassDistribution(i) {
CD = prior class distribution
For all properties prop of i
     Find parameter value v such that prop(i,v)
     For each class c
         CD[c] = CD[c] x P(v|c)
For all structural predicates struc involving i
     If struc is functional given i then
         Find the related individual j
         CD' = getClassDistribution(j)
         For each class c, CD[c] = CD[c] x CD'[c]
     Else /* non-determinate struc. pred. li/ms/ss */
         Find all related individuals J
         For each individual j of J
             CD''[j] = getClassDistribution(j)
         For each class c
             CD[c] = CD[c] x estimateP(li/ms/ss,CD'',c)
Return CD}
```

In order to deal with lists, multisets and sets, `estimateP(li/ms/ss,CD'',c)` applies the appropriate formula (Definitions 1,2,3) using `CD''[j][c]` as the probability of each element `j`.

Example 4. Here is a partial description of molecule `d1`:

```
class(d1,mutagenic).        atomch(d1_1,-0.117).
mol2atom(d1,d1_1).          ...
```

```
mol2atom(d1,d1_2).
...
atomel(d1_1,c).
...
atomty(d1_1,22).
...
```

```
bond2atom1(d1_1d1_2,d1_1).
bond2atom2(d1_1d1_2,d1_2).
kind(d1_1d1_2,7).
...
```

In order to classify molecule `d1` in mutagenesis, `1BC2` estimates its probability given each class using the properties of that individual, and the probabilities of individuals it is linked to. Since a molecule has no properties but the target predicate class, it considers the structural predicate: `mol2atom`. A molecule can have several atoms (`*:atom` in the ISP declaration of `mol2atom`) and P_{li} estimate (`li` at the end of the line) will be used (cf. Section 2). In order to apply P_{li}, `1BC2` requires the probability of each atoms of molecule `d1`: `d1_1`, `d1_2`, ... given a class `c`. Then the same algorithm is applied recursively! The probability of atom `d1_1` is estimated using its properties: `atomel(d1_1,c)`, `atomty(d1_1,22)`, and `atomch(d1_1,-0.117)`, and the probabilities of the individuals it is linked to: `bond2atom1(d1_1d1_2,d1_1)`, `bond2atom2(d1_6d1_1,d1_1)`, `bond2atom1(d1_1d1_7,d1_1)`. First for each class the a priori probability of that class is multiplied by the estimates of the probability of a carbon atom, the probability of an atomic number of 22, and the probability of a -0.117 charge for that class. Then it requires to estimate the probabilities of each bond, e.g. `d1_1d1_2`. Again the same algorithm is applied. In order to avoid to loop and come back to atoms from the bond, a limit on the number of times a structural predicate can be used is fixed by the user in the ISP declaration. Those limits are sensitive parameters. An accurate representation of individuals and consequently estimation of their probabilities depend on them. As illustrated by this example, a value of 1 prevents cycles when they are undesirable. When cycles are allowed, e.g. the tail predicate in lists, then the user could fix the limit to the maximal observed length. So here the probability of bond `d1_1d1_2` is estimated using its property: `kind(d1_1d1_2,7)`. We will detail in next paragraph how the probability of a bond being of kind `7` given each class is estimated. Those probabilities are then returned as the probabilities of bond `d1_1d1_2`. The same is done for all bonds in which atom `d1` occurs. Then, given those probabilities, P_{li} formula is used to get the probability of atom `d1_1` for each class. We will detail in next paragraph how τ is estimated. The same is done for all atoms of molecule `d1`. Finally P_{li} formula is used again to get the probability of the molecule given a list of its atoms.

The learning phase has to estimate the probabilities of empty lists, e.g. of atoms and of bonds, and the probabilities of each property, for each class. The current implementation relies on counting how many individuals satisfy each value of the property, taking into account the target predicate. For instance, it counts how many carbon atoms belong to mutagenic molecules, and how many carbon atoms belong to non-mutagenic molecules. It does the same for all parameters (`oxygen`, `hydrogen`, ...) of the `atomel`, and for all parameters of all other properties (atomic type, charge, kind of bond). The laplace estimate is used given those countings. The probability of the empty list is actually the parameter τ of a geometric distribution $P(l) = \tau(1-\tau)^l$ that has to be estimated given observed values: $l_1, l_2, \ldots, l_k$. We use a maximum likelihood estimates, i.e. τ that maximises the probabilities $p = P(l_1) \times P(l_2) \times \ldots \times P(l_k) = \tau(1-\tau)^{l_1} \ldots \tau(1-\tau)^{l_k}$.

It is the same as maximising $log(p) = \Sigma_i(log(\tau) + l_i \times log(1-\tau))$ or $\frac{log(p)}{k} = log(\tau) + \bar{l} \times log(1-\tau)$, where $\bar{l}$ is the average of l_i. The maximum necessarily corresponds to a zero of the derivative with respect to τ: $\frac{1}{t} - \bar{l} \times \frac{1}{1-\tau} = 0$. Therefore $\tau = \frac{1}{1+\bar{l}}$. So 1BC2 calculates the average number of atoms in molecules and the average number of bonds in atoms, for each class, in order to estimate respective probabilities of the empty lists. Of course, different estimates could be used.

4 Experiments

In this section we describe experimental results on several domains: Alzheimer's disease, mutagenesis and diterpene structure elucidation.

Experiments have been carried out with the list distribution (Definitions 1). The multiset distribution (Definition 2) has not been used since it gives the same results as the list distribution. Indeed, $P_{\rm ms}(s) = \frac{l!}{k_1!\ldots k_n!} P_{\rm li}(s)$ and $\frac{l!}{k_1!\ldots k_n!}$ does not depend on the class, so $argmax_c P_{\rm ms}(s|c) = argmax_c P_{\rm li}(s)$. The definition of $P_{\rm ss}$ involves a sum over all subsets. The number of subsets is exponential in the number of elements, therefore it is only practical in domains involving small sets, for instance sets in Alzheimer's disease involve a maximum of 9 elements, those in diterpenes contain 20 elements, but molecules in mutagenesis have up to 40 atoms. On Alzheimer's disease, we got very close, slightly lower, results for the subset distribution than for the list distribution. We intend to investigate this more fully in future work.

Experiments are evaluated both with accuracy and with ROC curves. *ROC curves* (ROC: Receiver Operating Characteristic) evaluate classifier performance in terms of *false positive rate* $FPr = \frac{FP}{TN+FP}$ (plotted on the X-axis) that needs to be minimized, and *true positive rate* $TPr = \frac{TP}{TP+FN}$ (plotted on the Y-axis) that needs to be maximized. The performance of a categorical classifier on a test set gives rise to a single point in ROC space. However, the performance of a probabilistic classifier on a test set gives rise to a ROC curve connecting (0,0) and (1,1) as described below. The area under this ROC curve (*AUC*) can be used as an evaluation metric. This metric is an aggregate of the expected performance of the classifier under different class and cost distributions.

In order to turn a probabilistic classifier into a categorical one, the decision rule used is usually 'if the predicted probability of being positive is larger than the predicted probability of being negative then predict positive else negative'. Equivalently, this corresponds to setting a fixed threshold 0.5 on the positive probability: if the positive probability is larger than this threshold we predict positive, else negative. A ROC curve can be constructed by varying this threshold from 1 (all predictions negative, corresponding to (0,0) in ROC space) to 0 (all predictions positive, corresponding to (1,1) in ROC space). This results in $n+1$ points in the ROC space, where n is the total number of examples. Equivalently, we can order all examples by decreasing predicted probability of being positive, and trace the ROC curve by starting in (0,0), stepping up when the example is actually positive and stepping to the right when it is negative, until we reach (1,1)[1]. The area under this ROC curve indicates the aggregated quality of all classifiers

[1] In the case of ties, we make the appropriate number of steps up and to the right at once, drawing a diagonal line segment.

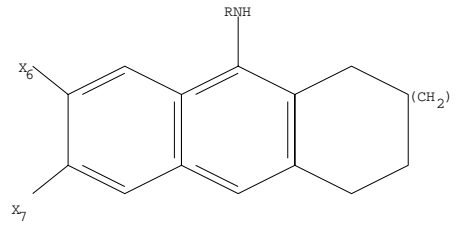


Fig. 2. Template of the tacrine molecule.

Table 1. Accuracy in the Alzheimer's disease domain.

Target	1BC	1BC2	Best rule inducer
Inhibit amine reuptake	67.7%	75.3%	86.1%
Low toxicity	74.4%	73.8%	81.9%
High acetyl cholinesterase inhibition	69.1%	68.7%	75.5%
Reversal of scopolamine-induced memory deficiency	62.2%	76.6%	61.0%

that can be constructed by setting different thresholds. Notice that the resulting curve is monotonic but not necessarily convex, especially not when it is constructed from a test set or in cross-validation.

4.1 Alzheimer's Disease

This dataset is about drugs against Alzheimer's disease [1]. It aims at comparing four desirable properties of such drugs:

- inhibit amine reuptake
- low toxicity
- high acetyl cholinesterase inhibition
- good reversal of scopolamine-induced memory deficiency

The aim is not to predict whether a molecule is good or bad, but rather whether the molecule is better or worse than another molecule for each of the four properties above. All molecules considered in this dataset have the structure of the tacrine molecule and they differ only by some substitutions on the ring structures (Figure 2).

New compounds are created by substituting chemicals for R, X_6 and X_7. The X substitution is represented by a substituent and its position. The R substitution consists of one or more alkyl groups linked to one or more benzene rings. The linkage can either be direct, or through N or O atoms, or through a CH bond. The R substitution is represented by its number of alkyl groups, a set of pairs of position and substituent, a number of ring substitutions, and a set of pairs of ring position and substituent.

Table 1 compares the accuracies of 1BC and 1BC2 on each of the four desirable properties. All experiments in this domain are carried out using a 10-fold cross validation. The last column shows the best accuracies reported by Boström and Asker using up-to-date rule inducers [1]. It should be noted that Boström and Asker applied several rule

Table 2. AUC in the Alzheimer's disease domain.

Target	1BC	1BC2
Inhibit amine reuptake	0.785	0.797
Low toxicity	0.816	0.805
High acetyl cholinesterase inhibition	0.780	0.773
Reversal of scopolamine-induced memory deficiency	0.680	0.649

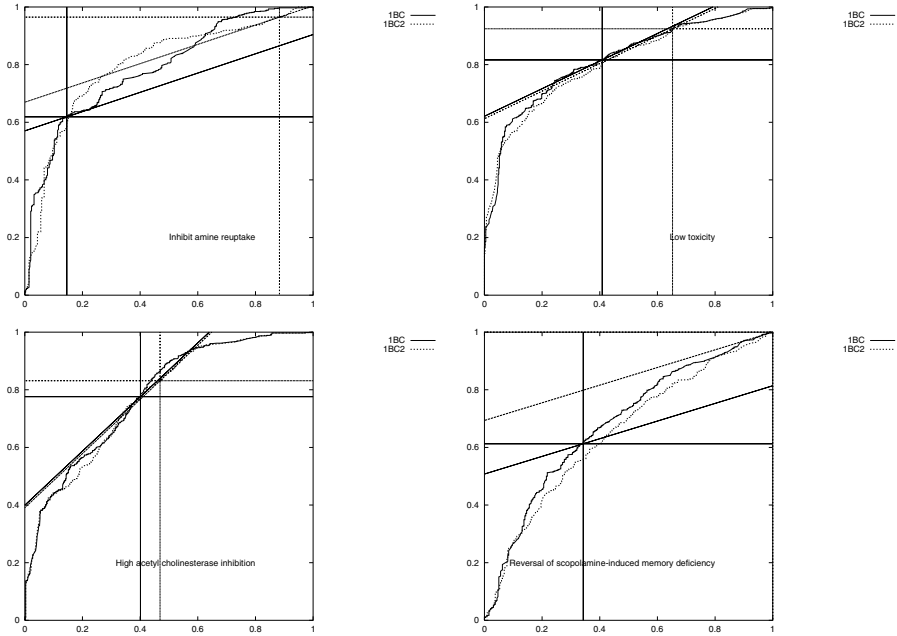


Fig. 3. ROC curves in the Alzheimer's disease domain for 1BC and 1BC2. The crosshairs denote the point chosen by the default probability threshold of 0.5, and the diagonal lines indicate iso-accuracy lines at those points (higher lines are better).

inducers, and that none of them got the best accuracy on all four target. Therefore those accuracies are surely an overestimate of the best overall rule inducer.

Table 2 compares the AUC values of 1BC and 1BC2 on the same four datasets. We see that the differences are very small. Figure 3 shows the corresponding ROC curves. Also indicated are: the points with probability threshold 0.5 corresponding to the accuracy results in Table 1 (the crosshairs), and the iso-accuracy lines through these points (the diagonal lines). From these ROC curves some interesting conclusions can be drawn. In the case of the second and third targets, the selected points on the curve by both classifiers are near-optimal – in fact, the curves have a whole flat segment which is near-optimal. For the first target, while 1BC2 doesn't select the best point, it does a much better job than 1BC, whose choice is clearly sub-optimal. Finally, for the fourth target 1BC2 again manages to choose the optimal point, which happens to be the majority class classifier – i.e., 1BC and all rule inducers performed worse than default!

Table 3. Accuracy in the mutagenesis domain.

Settings	1BC	1BC2	Progol	Regression
lumo and logp only	71.3%	71.3%		85%
lumo, logp, inda and ind 1 only	83.0%	83.0%		89%
Atoms and bonds only	80.3%	72.9%		
Plus lumo and logp	82.4%	72.9%	88%	
Plus inda and ind1	87.2%	72.9%	88%	

Table 4. AUC in the mutagenesis domain.

Target	1BC	1BC2
lumo and logp only	0.761	0.757
lumo, logp, inda and ind 1 only	0.895	0.895
Atoms and bonds only	0.857	0.861
Plus lumo and logp	0.862	0.862
Plus inda and ind1	0.904	0.863

Overall, the experiments on the Alzheimer's disease dataset suggest that 1BC2 is faster than 1BC (30 seconds compared to 1 minute), that the two produce fairly similar curves, and that 1BC2 does a much better job at determining a near-optimal point than 1BC. 1BC2 consistently does better on the positives and worse on the negatives than 1BC.

4.2 Mutagenesis

This problem concerns identifying mutagenic compounds [10,8]. We considered the "regression friendly" dataset. In these experiments, we used the atom and bond structure of the molecule as one setting, adding the lumo and logp properties to get a second setting, and finally adding boolean indicators I_a and I_1 as well. The latter four properties are propositional.

Table 3 reports the accuracy of different classifiers. The first two settings are propositional. Therefore, 1BC and 1BC2 get the same accuracy and almost identical ROC curves (Figure 4). In the relational settings, 1BC's is able to take advantage of the added propositional properties, while 1BC2 accuracy remains constant and significantly lower. This suggests that the trade-off between propositional and relational features chosen by 1BC2 leaves room for improvement. 1BC2 again chooses points on the curve which classify more instances as positive than 1BC, but on this domain this leads to worse accuracy results. The curves and AUC values are again fairly similar. It should be noted that 1BC runs the 10-fold cross validation in 5 minutes for relational settings while 1BC2 requires only 5 seconds!

4.3 Diterpene Structure Elucidation

The last dataset is concerned with Diterpenes, which are one of a few fundamental classes of natural products with about 5000 members known [2]. Structure elucidation

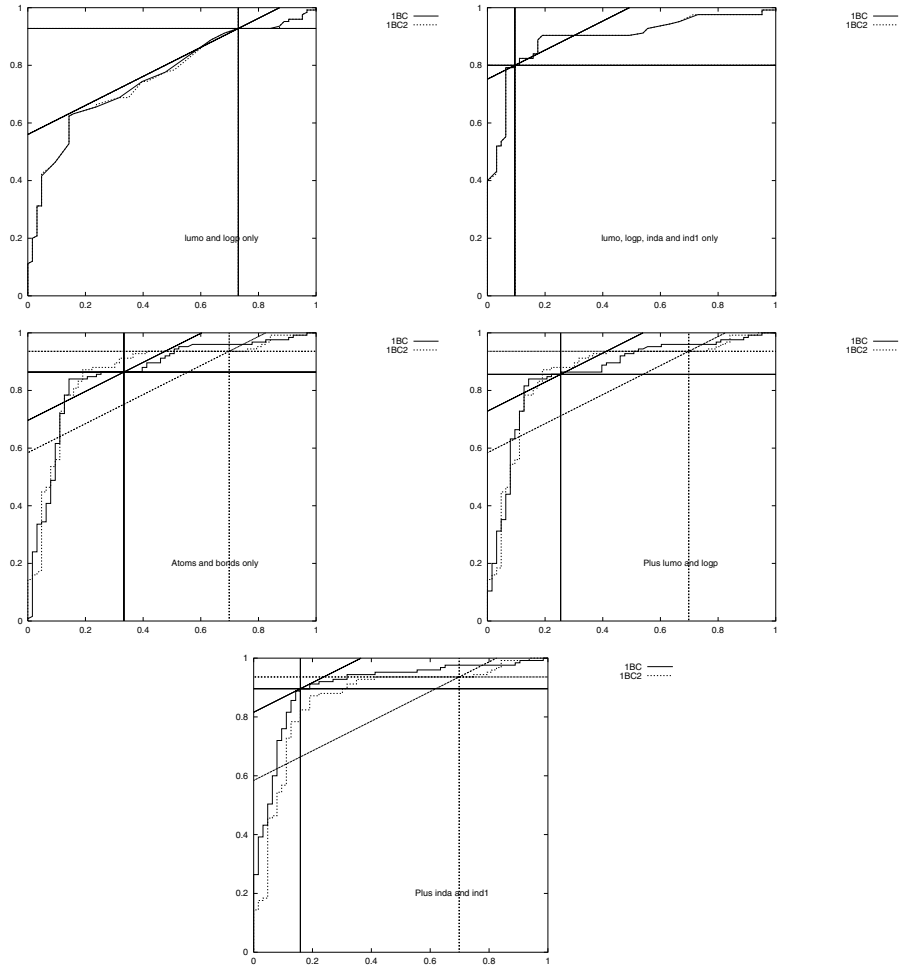


Fig. 4. ROC curves in the mutagenesis domain.

of diterpenes from C-NMR-Spectra (Nuclear magnetic Resonance) can be separated into three main stages: (1) identification of residues (ester and/or glycosides), (2) identification of the diterpene skeleton, and (3) arrangement of the residues on the skeleton. This dataset is concerned with the second stage, the identification of the skeleton. A skeleton is a unique connection of 20 carbon atoms, each with a specific atom number and, normalized to a pure skeleton molecule without residues, a certain multiplicity (s, d, t or q) measuring the number of hydrogens directly connected to a particular carbon atom: s stands for singulet, which means there is no proton (i.e., hydrogen) connected to the carbon; d stands for a doublet with one proton connected to the carbon; t stands for a triplet with two protons and q for a quartet with three protons bound to the carbon atom.

Table 5. Results in the diterpenes domain.

Settings	1BC	1BC2	C4.5	FOIL	RIBL
Propositional	77.6%	77.6%	78.5%	70.1%	79.0%
Relational	63.7%	55.1%		46.5%	86.5%
Propositional and relational	71.3%	66.5%		78.3%	91.2%

The data contain information on 1503 diterpenes with known structure. Two representations are available: a propositional one where the atom numbers are known, and a relational one without this information. In order to compare our results with [2], the accuracy is evaluated with a 3-fold cross-validation, and three settings were considered: each representation separately, and then together. We did not perform a ROC analysis on this domain because it has more than two classes (23 classes).

In this domain, 1BC and 1BC2 get a better accuracy than FOIL in the first two settings, but a worse accuracy than RIBL. In particular, they do not benefit from the combination of propositional and relational data as well as RIBL. At least they perform comparably to C4.5 on the propositional representation. This indicates a minimal property of a first-order learner, namely to perform as well as a propositional learner on propositional data. Comparing 1BC and 1BC2, while 1BC2's accuracy is slightly lower than 1BC's, 1BC2's run time is 20 minutes and 1BC's is 6 hours.

5 Concluding Remarks

Our previous system 1BC is a first-order naive Bayesian classifier that generates a set of first-order features that are then used dynamically as attributes of a propositional naive Bayesian classifier [3]. 1BC2 is a first-order naive Bayesian classifier too. However, it is much more difficult to conceive an equivalent algorithm consisting of a propositionalisation, followed by some propositional naive Bayesian classifier. For this reason, it is a true first-order Bayesian classifier.

In order to properly upgrade the attribute-value naive Bayes assumption to first-order objects, we first defined probability distributions over lists, multisets and sets that allow us to estimate the probability of a list (resp. a multiset and a set) of elements given the probability of those elements. The principle of 1BC2 to classify an structured individual is then to consider what its top-level structure is (tuple, list, set or multiset of elements) and then to estimate the probability of this structure from the probability of its elements. If an element is a structured object itself, its probability is estimated using the same principle. The probabilities of non-structured objects are estimated using Laplace estimate from countings done during the learning phase.

We evaluated the performance of 1BC2 on three datasets. On the first two datasets, a ROC analysis showed that 1BC and 1BC2 obtain comparable ROC curves, but that the default 0.5 probability thresholds lead 1BC2 to classify more instances as positive than 1BC. While on Alzheimer's disease this led to comparable or significantly better accuracy, on mutagenesis this situation was reversed. On the third dataset 1BC2 obtained considerably worse accuracy than 1BC when relational features were involved. This

suggests that the trade-off between propositional and relational features chosen by 1BC2 leaves room for improvement, something we intend to investigate in future work.

The main strength of 1BC2 lies in its simplicity: since 1BC2 does not consider complex hypotheses, its learning time is considerably reduced. In this respect, 1BC2 is clearly quicker than 1BC: while 1BC has to generate first-order features at learning time, and then to match them at classification time, 1BC2 just go through the structure of each individual, to count at learning time, and to estimate probabilities according to appropriate distributions at classification time. Their run times on structured data differ roughly of one order of magnitude: minutes vs. seconds in mutagenesis, hours vs. minutes for diterpenes.

We have also found in our experiments that 1BC2 behaved identically, regardless of whether the list or the subset distribution was chosen. At present we do not know whether this was a coincidence on the domains considered, or whether they are genuinely equivalent when used in a naive Bayesian classifier. This will be investigated in future work. Further perspectives for future work include a non-exponential formula for the subset distribution, further investigations on estimates of non-structured and structured objects, in particular to get the best accuracy of 1BC and 1BC2 at the speed of 1BC2, and finally the use of those new estimators in other applications, for instance Support Vector Machines [5].

Acknowledgements

Thanks are due to Henrik Boström and Sašo Džeroski for providing us with the Alzheimer and Diterpene datasets, respectively. Part of this work is supported by the IST project IST-1999-11495 *Data Mining and Decision Support for Business Competitiveness: Solomon Virtual Enterprise*.

References

1. H. Boström and L. Asker. Combining divide-and-conquer and separate-and-conquer for efficient and effective rule induction. In S. Džeroski and P. Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 33–43. Springer-Verlag, 1999.
2. Sasŏ Džeroski, Steffen Schulze-Kremer, Karsten R. Heidtke, Karsten Siems, Dietrich Wettschereck, and Hendrik Blockeel. Diterpene structure elucidation from ^{1}3c nmr spectra with inductive logic programming. *Applied Artificial Intelligence*, 12(5):363–383, July-August 1998. Special Issue on First-Order Knowledge Discovery in Databases.
3. P. Flach and N. Lachiche. 1BC: A first-order Bayesian classifier. In S. Džeroski and P. Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 92–103. Springer-Verlag, 1999.
4. Jerome H. Friedman. On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, March 1997.
5. Thomas Gärtner and Peter Flach. A linear kernel on strongly typed terms. In *Multi-Relational Data Mining workshop, Joint workshop of ECML'01 and PKDD'01*, 2001.
6. Nicolas Lachiche and Peter A. Flach. A first-order representation for knowledge discovery and bayesian classification on relational data. In Pavel Brazdil and Alipio Jorge, editors, *Data*

Mining, decision Support, Meta-learning and ILP : Forum for Practical Problem Presentation and Prospective Solutions (DDMI-2000), Workshop of 4th International Conference on Principles of Data Mining and Knowledge Discovery (PKDD-2000), pages 49–60, Lyon, September 2000.

7. N. Lavrač, S. Džeroski, and M. Grobelnik. Learning nonrecursive definitions of relations with LINUS. In Y. Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 265–281. Springer-Verlag, 1991.
8. S. Muggleton, A. Srinivasan, R. King, and M. Sternberg. Biochemical knowledge discovery using Inductive Logic Programming. In H. Motoda, editor, *Proceedings of the first Conference on Discovery Science*, Berlin, 1998. Springer-Verlag.
9. U. Pompe and I. Kononenko. Naive Bayesian classifier within ILP-R. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 417–436. Department of Computer Science, Katholieke Universiteit Leuven, 1995.
10. A. Srinivasan, S. Muggleton, R.D. King, and M.J.E. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 217–232. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

RSD: Relational Subgroup Discovery through First-Order Feature Construction

Nada Lavrač[1], Filip Železný[2], and Peter A. Flach[3]

[1] Institute Jožef Stefan, Ljubljana, Slovenia
nada.lavrac@ijs.si
[2] Czech Technical University, Prague, Czech Republic
zelezny@fel.cvut.cz
[3] University of Bristol, Bristol, UK
peter.flach@bristol.ac.uk

Abstract. Relational rule learning is typically used in solving classification and prediction tasks. However, relational rule learning can be adapted also to subgroup discovery. This paper proposes a propositionalization approach to relational subgroup discovery, achieved through appropriately adapting rule learning and first-order feature construction. The proposed approach, applicable to subgroup discovery in individual-centered domains, was successfully applied to two standard ILP problems (East-West trains and KRK) and a real-life telecommunications application.

1 Introduction

Developments in *descriptive induction* have recently gained much attention. These involve mining of association rules (e.g., the APRIORI association rule learning algorithm [1]), subgroup discovery (e.g., the MIDOS subgroup discovery algorithm [22]), symbolic clustering and other approaches to non-classificatory induction.

The methodology presented in this paper can be applied to relational subgroup discovery. As in the MIDOS approach, a subgroup discovery task can be defined as follows: given a population of individuals and a property of those individuals we are interested in, find population subgroups that are statistically 'most interesting', e.g., are as large as possible and have the most unusual statistical (distributional) characteristics with respect to the property of interest. This paper aims at solving a slightly modified subgroup discovery task that can be stated as follows. Again, the input is a population of individuals and a property of those individuals we are interested in, and the output are population subgroups that are statistically 'most interesting': are as large as possible, have the most unusual statistical (distributional) characteristics with respect to the property of interest and are sufficiently distinct for detecting most of the target population.

Notice an important aspect of the above two definitions. In both, there is a predefined property of interest, meaning that both aim at characterizing popu-

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 149–165, 2003.

lation subgroups of a given *target* class. This property indicates that rule learning may be an appropriate approach for solving the task. However, we argue that standard propositional rule learning [6,16] and relational rule learning algorithms [19] are unsuitable for subgroup discovery. The main drawback is the use of the covering algorithm for rule set construction. Only the first few rules induced by a covering algorithm may be of interest as subgroup descriptions with sufficient coverage, thus representing a 'chunk' of knowledge characterizing a sufficiently large population of covered examples. Subsequent rules are induced from smaller and strongly biased example subsets, i.e., subsets including only positive examples not covered by previously induced rules. This bias prevents a covering algorithm to induce descriptions uncovering significant subgroup properties of the entire population. A remedy to this problem is the use of a weighted covering algorithm, as demonstrated in this paper, where subsequently induced rules with high coverage allow for discovering interesting subgroup properties of the entire population.

This paper investigates how to adapt classification rule learning approaches to subgroup discovery, by exploiting the information about class membership in training examples. This paper shows how this can be achieved by appropriately modifying the covering algorithm (weighted covering algorithm) and the search heuristics (weighted relative accuracy heuristic). The main advantage of the proposed approach is that each rule with high weighted relative accuracy represents a 'chunk' of knowledge about the problem, due to the appropriate tradeoff between accuracy and coverage, achieved through the use of the weighted relative accuracy heuristic.

The paper is organized as follows. In Section 2 the background for this work is explained: propositionalization through first-order feature construction, irrelevant feature elimination, the standard covering algorithm used in rule induction, the standard heuristics as well as the weighted relative accuracy heuristic, probabilistic classification and rule evaluation in the ROC space. Section 3 presents the proposed relational subgroup discovery algorithm. Section 4 presents the experimental evaluation in two standard ILP problems (East-West trains and KRK) and a real-life telecommunications application. Section 5 concludes by summarizing the results and presenting plans for further work.

2 Background

This section presents the backgrounds: propositionalization through first-order feature construction, irrelevant feature elimination, the standard covering algorithm used in rule induction, the standard heuristics as well as the weighted relative accuracy heuristic, probabilistic classification and rule evaluation in the ROC space.

2.1 Propositionalization through First-Order Feature Construction

The background knowledge used to construct hypotheses is a distinctive feature of relational rule learning (and inductive logic programming, in general). It is

well known that relevant background knowledge may substantially improve the results of learning in terms of accuracy, efficiency, and the explanatory potential of the induced knowledge. On the other hand, irrelevant background knowledge will have just the opposite effect. Consequently, much of the art of inductive logic programming lies in the appropriate selection and formulation of background knowledge to be used by the selected ILP learner.

By devoting enough effort to the construction of features, to be used as background knowledge in learning, even complex relational learning tasks can be solved by simple propositional rule learning systems. In propositional learning, the idea of augmenting an existing set of attributes with new ones is known under the term constructive induction. A first-order counterpart of constructive induction is predicate invention. This work takes the middle ground: we perform a simple form of predicate invention through first-order feature construction, and use the constructed features for relational rule learning, which thus becomes propositional.

Our approach to first-order feature construction can be applied in the so-called *individual-centered* domains, where there is a clear notion of individual, and learning occurs at the level of individuals only. Such domains include classification problems in molecular biology, for example, where the individuals are molecules. Often, individuals are represented by a single variable, and the target predicates are either unary predicates concerning boolean properties of individuals, or binary predicates assigning an attribute-value or a class-value to each individual. It is however also possible that individuals are represented by tuples of variables.

In our approach to first-order feature construction, described in [9,12,10], local variables referring to parts of individuals are introduced by so-called *structural predicates*. The only place where nondeterminacy can occur in individual-centered representations is in structural predicates. Structural predicates introduce new variables. In a given language bias for first-order feature construction, a first-order feature is composed of one or more structural predicates introducing a new variable, and of *utility predicates* as in LINUS [13] (called *properties* in [9]) that 'consume' all new variables by assigning properties of individuals or their parts, represented by variables introduced so far. Utility predicates do not introduce new variables.

Individual-centered representations have the advantage of a strong language bias, because local variables in the bodies of rules either refer to the individual or to parts of it. However, not all domains are amenable to the approach we use in this paper - in particular, we cannot learn recursive clauses, and we cannot deal with domains where there is not a clear notion of individual (e.g., many program synthesis problems).

2.2 Irrelevant Feature Elimination

Let L denote the set of all features constructed by a first-order feature construction algorithm. Some features defined by the language bias may be irrelevant for the given learning task. Irrelevant features can be detected and eliminated in

preprocessing. Besides reducing the hypothesis space and facilitating the search for the solution, the elimination of irrelevant features may contribute to a better understanding of the problem domain.

It can be shown that if a feature $l' \in L$ is irrelevant then for every complete and consistent hypothesis $H = H(E, L)$, built from example set E and feature set L whose description includes feature l', there exists a complete and consistent hypothesis $H' = H(E, L')$, built from the feature set $L' = L \backslash \{l'\}$ that excludes l'. This theorem is the basis of an irrelevant feature elimination algorithm proposed in [15].

Note that usually the term feature is used to denote a positive literal (or a conjunction of positive literals; let us, for the simplicity of the arguments below, assume that a feature is a single positive literal). In the hypothesis language, the existence of one feature implies the existence of two complementary literals: a positive and a negated literal. Since each feature implies the existence of two literals, the necessary and sufficient condition for a feature to be eliminated as irrelevant is that both of its literals are irrelevant.

This observation directly implies the approach taken in this work. First we convert the starting feature set to the corresponding literal set which has twice as many elements. After that, we eliminate the irrelevant literals and, in the third step, we construct the reduced set of features which includes all the features which have at least one of their literals in the reduced literal set.

It must be noted that direct detection of irrelevant features (without conversion to and from the literal form) is not possible except in the trivial case where two (or more) features have identical values for all training examples. Only in this case a feature f exists whose literals f and $\neg f$ cover both literals g and $\neg g$ of some other feature. In a general case if a literal of feature f covers some literal of feature g then the other literal of feature g is not covered by the other literal of feature f. But it can happen that this other literal of feature g is covered by a literal of some other feature h. This means that although there is no such feature f that covers both literals of feature g, feature g can still turn out to be irrelevant.

2.3 Rule Induction Using the Covering Algorithm

Rule learning typically consists of two main procedures: the search procedure that performs search in order to find a single rule and the control procedure that repeatedly executes the search. In the propositional rule learner CN2 [5,6], for instance, the search procedure performs beam search using classification accuracy of the rule as a heuristic function. The accuracy of rule $H \leftarrow B$ is equal to the conditional probability of head H, given that the body B is satisfied: $Acc(H \leftarrow B) = p(H|B)$.

The accuracy measure can be replaced by the weighted relative accuracy, defined in Equation 1. Furthermore, different probability estimates, like the Laplace [4] or the m-estimate [3,7], can be used for estimating the above probability and the probabilities in Equation 1.

Additionally, a rule learner can apply a significance test to the induced rule. The rule is considered to be significant, if it locates a regularity unlikely to have occurred by chance. To test significance, for instance, CN2 uses the likelihood ratio statistic [6] that measures the difference between the class probability distribution in the set of examples covered by the rule and the class probability distribution in the set of all training examples. Empirical evaluation in [4] shows that applying a significance test reduces the number of induced rules (but also slightly degrades the predictive accuracy).

Two different control procedures are used in CN2: one for inducing an ordered list of rules and the other for the unordered case. When inducing an ordered list of rules, the search procedure looks for the best rule, according to the heuristic measure, in the current set of training examples. The rule predicts the most frequent class in the set of examples, covered by the induced rule. Before starting another search iteration, all examples covered by the induced rule are removed. The control procedure invokes a new search, until all the examples are covered.

In the unordered case, the control procedure is iterated, inducing rules for each class in turn. For each induced rule, only covered examples belonging to that class are removed, instead of removing all covered examples, like in the ordered case. The negative training examples (i.e., examples that belong to other classes) remain and positives are removed in order to prevent CN2 finding the same rule again.

2.4 The Weighted Relative Accuracy Heuristic

Weighted relative accuracy can be meaningfully applied both in the descriptive and predictive induction framework; in this paper we apply this heuristic for subgroup discovery.

We use the following notation. Let $n(B)$ stand for the number of instances covered by rule $H \leftarrow B$, $n(H)$ stand for the number of examples of class H, and $n(H.B)$ stand for the number of correctly classified examples (true positives). We use $p(H.B)$ etc. for the corresponding probabilities. We then have that rule accuracy can be expressed as $Acc(H \leftarrow B) = p(H|B) = \frac{p(H.B)}{p(B)}$. Weighted relative accuracy [14,21], a reformulation of one of the heuristics used in MIDOS [22], is defined as follows.

$$WRAcc(H \leftarrow B) = p(B).(p(H|B) - p(H)). \tag{1}$$

Weighted relative accuracy consists of two components: generality $p(B)$, and relative accuracy $p(H|B) - p(H)$. The second term, relative accuracy, is the accuracy gain relative to the fixed rule $H \leftarrow true$. The latter rule predicts all instances to satisfy H; a rule is only interesting if it improves upon this 'default' accuracy. Another way of viewing relative accuracy is that it measures the utility of connecting rule body B with a given rule head H. However, it is easy to obtain high relative accuracy with highly specific rules, i.e., rules with low generality $p(B)$. To this end, generality is used as a 'weight', so that weighted relative accuracy trades off generality of the rule ($p(B)$, i.e., rule coverage) and relative accuracy ($p(H|B) - p(H)$).

2.5 Probabilistic Classification

The induced rules can be ordered or unordered. Ordered rules are interpreted as a decision list [20] in a straight-forward manner: when classifying a new example, the rules are sequentially tried and the first rule that covers the example is used for prediction.

In the case of unordered rule sets, the distribution of covered training examples among classes is attached to each rule. Rules of the form:

$$H \leftarrow B \ [ClassDistribution]$$

are induced, where numbers in the *ClassDistribution* list denote, for each individual class H, how many training examples of this class are covered by the rule. When classifying a new example, all rules are tried and those covering the example are collected. If a clash occurs (several rules with different class predictions cover the example), a voting mechanism is used to obtain the final prediction: the class distributions attached to the rules are summed to determine the most probable class.

2.6 Area Under the ROC Curve Evaluation

A point on the *ROC curve*[1] shows classifier performance in terms of false alarm or *false positive rate* $FPr = \frac{FP}{TN+FP}$ (plotted on the X-axis) that needs to be minimized, and sensitivity or *true positive rate* $TPr = \frac{TP}{TP+FN}$ (plotted on the Y-axis) that needs to be maximized. In the ROC space, an appropriate tradeoff, determined by the expert, can be achieved by applying different algorithms, as well as by different parameter settings of a selected data mining algorithm.

3 Relational Subgroup Discovery

We have devised a relational subgroup discovery system RSD on principles that employ the following main ingredients: exhaustive first-order feature construction, elimination of irrelevant features, implementation of a relational rule learner, use of the weighted covering algorithm and incorporation of example weights into the weighted relative accuracy heuristic.

The input to RSD consists of

- a relational database (further called *input data*) containing one main table (relation) where each row corresponds to a unique *individual* and one attribute of the main table is specified as the *class* attribute, and
- a mode-language definition used to construct first-order features.

The main output of RSD is a set of subgroups whose class-distributions differ substantially from those of the complete data-set. The subgroups are identified by conjunctions of symbols of pre-generated first-order features. As a by-product,

[1] ROC stands for Receiver Operating Characteristic [11,18]

RSD also provides a file containing the mentioned set of features and offers to export a single relation (as a text file) with rows corresponding to individuals and fields containing the truth values of respective features for the given individual. This table is thus a propositionalized representation of the input data and can be used as an input to various attribute-value learners.

3.1 RSD First-Order Feature Construction

The design of an algorithm for constructing first-order features can be split into three relatively independent problems.

- Identifying all first-order literal conjunctions that by definition form a feature (see [9], briefly described in Section 2.1), and at the same time comply to user-defined constraints (mode-language). Such features do not contain any constants and the task can be completed independently of the input data.
- Extending the feature set by variable instantiations. Certain features are copied several times with some variables substituted to constants 'carefully' chosen from the input data.
- Detecting irrelevant features (see [15], briefly described in Section 2.2) and generating propositionalized representations of the input data using the generated feature set.

Identifying Features. Motivated by the need to easily recycle language-bias declarations already present for numerous ILP problems, RSD accepts declarations very similar to those used by the systems Aleph [2] or Progol [17], including variable typing, moding, setting a *recall* parameter etc, used to syntactically constrain the set of possible features. For example, a structural predicate declaration in the well-known domain of East-West trains would be such as

```
:-modeb(1, hasCar(+train, -car)).
```

where the recall number 1 determines that a feature can address at most one car of a given train. Property predicates are those with no output variables (labeled with the minus sign). The head declaration always contains exactly one variable of the input mode (e.g. `+train` in our example).

Various settings such as the maximal *feature length* denoting the maximal number of literals allowed in a feature, maximal *variable depth* etc. can also be specified, otherwise their default value is assigned.

RSD will produce the exhaustive set of features satisfying the mode and setting declarations. No feature produced by RSD can be decomposed into a conjunction of two features.

Employing Constants. As opposed to Aleph or Progol declarations, RSD does not use the # sign to denote an argument which should be a constant. In the mentioned systems, constants are provided by a single saturated example, while

RSD extract constants from the whole input data. The user can instead utilize the reserved property predicate `instantiate/1`, which does not occur in the background knowledge, to specify a variable which should be substituted with a constant. For example, out of the following declarations

```
:-modeh(1, train(+train)).
:-modeb(1, hasCar(+train, -car)).
:-modeb(1, hasLoad(+car, -load)).
:-modeb(1, hasShape(+load, -shape).
:-modeb(3, instantiate(+shape)).
```

exactly one feature will be generated[2]:

```
f1(A) :- hasCar(A,B),hasLoad(B,C),hasShape(C,D),instantiate(D).
```

However, in the second step, after consulting the input data, `f1` will be substituted by 3 features (due to the recall parameter 3 in the last declaration) where the `instantiate/1` literal is removed and the `D` variable is substituted with 3 most frequent constants out of all values of `D` which make the body of `f1` provable in the input data. One of them will be

```
f1(A) :- hasCar(A,B),hasLoad(B,C),hasShape(C,rectangle).
```

Filtering and Applying Features. RSD implements the simplest scheme of feature filtering from [15]. This means that features with complete of empty coverage on the input data will be retracted from the feature set. In the current implementation, RSD does not check for irrelevance of a feature caused by the presence of other features. As the product of this third, final step of feature construction, the system exports an attribute representation of the input data based on the truth values of respective features, in a file of parametrizable format.

3.2 RSD Rule Induction Algorithm

A part of RSD is a subgroup discovery program which can accept data propositionalized by the feature constructor described above. The algorithm acquires some basic principles of the CN2 rule learner [6], which are however adapted in several substantial ways to meet the interests of subgroup discovery. The principal modifications are outlined below.

The Weighted Covering Algorithm. In the classical covering algorithm for rule set induction, only the first few induced rules may be of interest as subgroup

[2] Strictly speaking, the feature is solely the *body* of the listed clause. However, clauses such as the one listed will also be called features in the following as this will cause no confusion.

descriptors with sufficient coverage, since subsequently induced rules are induced from biased example subsets, i.e., subsets including only positive examples not covered by previously induced rules. This bias constrains the population for subgroup discovery in a way that is unnatural for the subgroup discovery process which is, in general, aimed at discovering interesting properties of subgroups of the entire population. In contrast, the subsequent rules induced by the proposed weighted covering algorithm allow for discovering interesting subgroup properties of the entire population.

The weighted covering algorithm performs in such a way that covered positive examples are not deleted from the current training set. Instead, in each run of the covering loop, the algorithm stores with each example a count how many times (with how many rules induced so far) the example has been covered. Weights derived from these example counts then appear in the computation of *WRAcc*. Initial weights of all positive examples e_j equals $w(e_j, 0) = 1$. Weights of covered positive examples decrease according to the formula $w(e_j, i) = \frac{1}{i+1}$, where $w(e_j, i)$ is the weight of example e_j being covered i times[3].

Modified WRAcc Heuristic with Example Weights. The modification of CN2 reported in [21] affected only the heuristic function: weighted relative accuracy was used as search heuristic, instead of the accuracy heuristic of the original CN2, while everything else stayed the same. In this work, the heuristic function was further modified to enable handling example weights, which provide the means to consider different parts of the instance space in each iteration of the weighted covering algorithm.

In the *WRAcc* computation (Equation 1) all probabilities are computed by relative frequencies. An example weight measures how important it is to cover this example in the next iteration. The initial example weight $w(e_j, 0) = 1$ means that the example hasn't been covered by any rule, meaning 'please cover this example, it hasn't been covered before', while lower weights mean 'don't try too hard on this example'. The modified *WRAcc* measure is then defined as follows

$$WRAcc(H \leftarrow B) = \frac{n'(B)}{N'}\left(\frac{n'(H.B)}{n'(B)} - \frac{n'(H)}{N'}\right). \tag{2}$$

where N' is the sum of the weights of all examples, $n'(B)$ is the sum of the weights of all covered examples, and $n'(H.B)$ is the sum of the weights of all correctly covered examples.

To add a rule to the generated rule set, the rule with the maximum *WRAcc* measure is chosen out of those rules in the search space, which are not yet present in the rule set produced so far (all rules in the final rule set are thus distinct, without duplicates).

[3] Whereas this approach is referred to as *additive*, another option is the *multiplicative* approach, where, for a given parameter $\gamma < 1$, weights of covered examples decrease as follows: $w(e_j, i) = \gamma^i$. Note that the weighted covering algorithm with $\gamma = 1$ would result in finding the same rule over and over again, whereas with $\gamma = 0$ the algorithm would perform the same as the standard CN2 algorithm.

Probabilistic Classification and Area under the ROC Curve Evaluation. The method, which is used for evaluation of predictive performance of the subgroup discovery results employs the combined probabilistic classifications of all subgroups (a set of rules as a whole). If we always choose the most likely predicted class, this corresponds to setting a fixed threshold 0.5 on the positive probability: if the positive probability is larger than this threshold we predict positive, else negative. A ROC curve can be constructed by varying this threshold from 1 (all predictions negative, corresponding to (0,0) in the ROC space) to 0 (all predictions positive, corresponding to (1,1) in the ROC space). This results in $n+1$ points in the ROC space, where n is the total number of examples. Equivalently, we can order all examples by decreasing the predicted probability of being positive, and tracing the ROC curve by starting in (0,0), stepping up when the example is actually positive and stepping to the right when it is negative, until we reach (1,1)[4]. The area under this ROC curve indicates the combined quality of all subgroups. This method can be used with a test set or in cross-validation: the resulting curve is not necessarily convex, but the area under the curve still indicates the quality of the combined subgroups for probabilistic prediction[5].

4 Experimental Evaluation

4.1 Materials

We have performed experiments with RSD on two popular ILP data sets: the King-Rook-King illegal chess endgame positions (KRK) and East-West trains.

We applied RSD also to a real-life problem in telecommunications. The data (described in detail in [23]) represent incoming calls to an enterprise, which were transferred to a particular person by the telephone receptionist. The company has two rather independent divisions ('datacomm', 'telecomm') and people in each of them may have a technical or commercial role. These two binary divisions define four classes of incoming calls ('data_tech', 'data_comm', 'tele_tech', 'tele_comm'), depending on the person the call was redirected to. The fifth class ('other') labels the calls (of smaller interest) going to other employees than those mentioned. The problem is to define subgroups of incoming calls usually falling into a given class. The motivation for subgroup discovery in this domain is to select people within a given class that can substitute each other in helping the caller.

Table 1 lists the basic properties of the experimental data.

4.2 Procedures

We have applied the RSD algorithm with each of the data sets in the following manner.

[4] In the case of ties, we make the appropriate number of steps up and to the right at once, drawing a diagonal line segment.

[5] A description of this method applied to decision tree induction can be found in [8].

Table 1. Basic properties of the experimental data.

Domain	Individual	No. of examples	No. of classes
KRK	KRK position	1000	2
Trains	Train	20	2
Telecommunication	Incoming call	1995	5

Table 2. The language bias and number of generated features for each of the experimental domain.

	KRK	Trains	Tele
No. of body declarations	10	16	11
Max. clause lenght	3	8	8
Max var. depth	2	4	4
No. of features before instantiations	42	219	4
No. of features after instantiations	42	2429	3729

- First, we generated a set of features and, except for the KRK domain, we expanded the set with features containing variable instantiations.
- Then the feature sets were used to produce a propositional representation of each of the data sets.
- From propositional data we induced rules with the RSD subgroup discovery algorithm as well as with the standard coverage approach.
- Finally, we compared the pairs of rule sets in terms of properties that are of interest for subgroup discovery.
- In one domain we evaluated the induced rule sets also from the point of view of predictive classification. To do so, we performed a classification test on the unseen part of the data, where rules were interpreted by the method of probabilistic classification, suing a ROC plot to evaluate the results.

4.3 Results

Feature Construction. Table 2 shows the values of the basic parameters of the language-bias declarations provided to the feature constructor for each given domain as well as the number of generated features before and after the expansion due to variable instantiations.

Table 3 below presents a few examples of features generated for each of the three domains together with a verbal interpretation.

Subgroup Discovery. The following experiment compares the results of the search for interesting subgroups by means of four techniques described in detail in the previous sections:

- Standard covering algorithm with rule accuracy as a selection measure (Cov + Acc).
- Example weighting (i.e., weighted covering algorithm) with rule accuracy as a selection measure (We + Acc).

Table 3. Examples of generated features.

KRK	`f6(A):-king1_rank(A,B),rook_rank(A,C),adj(C,B).`
meaning	first king's rank adjacent to rook's rank
Trains	`f5(A):-hasCar(A,B),carshape(B,ellipse),carlength(B,short).`
meaning	has a short elliptic car
Tele	`f3(A):-call_date(A,B),day_is(B,mon).`
meaning	call accepted on Monday
	`f301(A):-ext_number(A,B),prefix(B,[0,4,0,7]).`
meaning	caller's number starts with 0407

Table 4. Average values of significance and coverage of the resulting rules of subgroup discovery conducted by four different combinations of techniques.

KRK	Cov + WRAcc	We + WRAcc	Cov + Acc	We + Acc
Avg. Significance	10.48	15.23	8.76	11.80
Avg. Coverage	16.84%	17.11%	8.56%	14.97%
Trains	Cov + WRAcc	We + WRAcc	Cov + Acc	We + Acc
Avg. Significance	1.49	1.69	0.60	1.28
Avg. Coverage	26.66%	17.00%	6.67%	14.16%
Tele	Cov + WRAcc	We + WRAcc	Cov + Acc	We + Acc
Avg. Significance	9.92	11.55	3.17	4.87
Avg. Coverage	5.78%	3.2%	0.0012%	0.0012%

- Standard covering algorithm with weighted relative accuracy as a selection measure (Cov + WRAcc).
- Example weighting with weighted relative accuracy as a selection measure (We + WRAcc).

The resulting rule sets are evaluated on the basis of average rule *significance* and average rule coverage. The significance of a rule is measured as in the CN2 algorithm [6], i.e., as the value of

$$\sum_i f_i \log \frac{f_i}{p_i} \tag{3}$$

where for each class i, p_i denotes the number of instances classified into i within the whole data set, and f_i denotes the number of such classified instances in the subset, where the rule's body holds true.

Table 4 shows the average significance and coverage values of rules produced in the three respective domains by each of the four combination of techniques, while Figure 1 shows the significance values for all individual rules.

By making pairwise comparisons we can investigate the influence of replacing the standard covering algorithm (*Cov*) by the weighted covering algorithm *We*), and the affects of replacing the standard accuracy heuristic (*Acc*) with the weighted relative accuracy heuristic (*WRAcc*). The following observations can be made from the quantitative results.

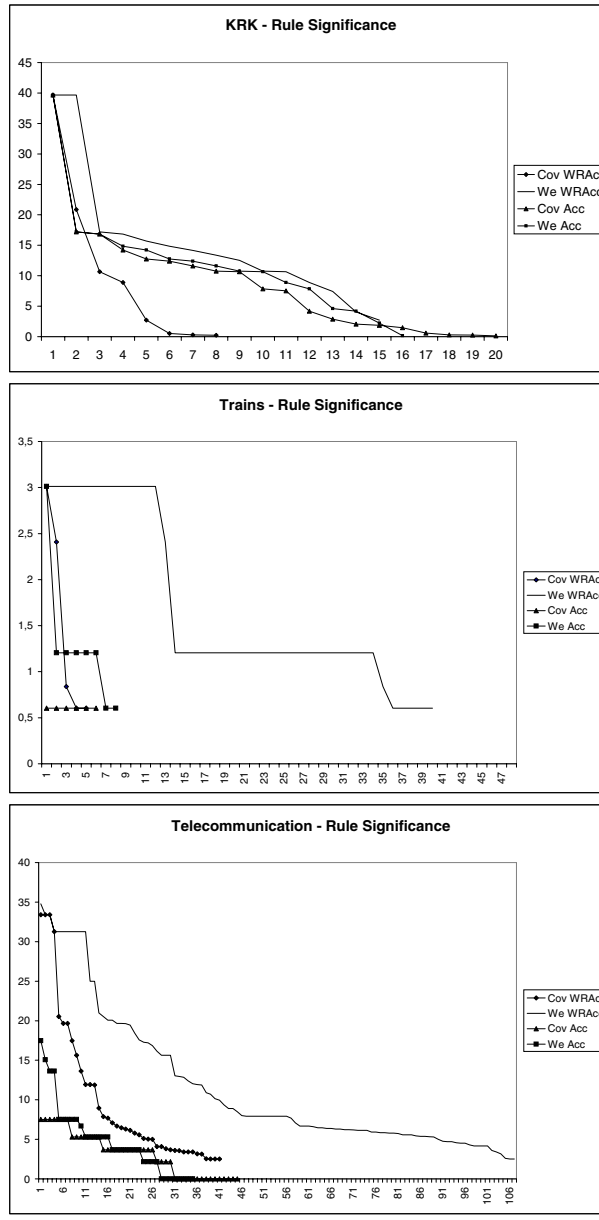


Fig. 1. The significance of individual rules produced by each of the four combinations of techniques in the three data domains. Rules are sorted decreasingly by their significance value. Note that the number of rules produced by the example weighting algorithm (*We*) is parameterizable by the setting of the weight-decay rate and the total weight threshold for stopping the search. We used the same threshold for all *We*-based experiments.

- *Example weighting increases* the average rule *significance* with respect to the standard covering algorithm for both cases of combination with the accuracy or weighted relative accuracy measure, in all three data domains. This follows from Table 4 above as well as from Figure 1.
- The use of *weighted relative accuracy* heuristic *increases* the average rule *coverage* with respect to the case of using the accuracy measure. This holds for both cases of combination with the standard covering algorithm or the technique of example weighting, in all three data domains. This follows from Table 4 (it is particularly striking in the Telecommunication domain).
- A consequence of the previous observation is that in the standard covering algorithm, the use of *weighted relative accuracy* leads to smaller output rule sets. This can be verified in Figure 1.
- It may seem surprising that the combination Cov + WRAcc yields a higher average rule coverage than We + WRAcc for Trains and Telecommunication. However, note that it does not make sense to compare coverage results across Cov/We. The reason is that the rule generation process using We (gradually tending to produce more specific rules) is stopped at an instant dictated by the user-specified threshold parameter for the sum of all examples' weights. Thus the coverage values are to be compared between methods both using Cov, and between methods both using We with the same threshold value setting.

Classification. Although the aim of RSD is to merely discover rules representing interesting subgroups, such rules may as well be used for classification purposes. For a binary classification problem, we can interpret the rule set as explained in Section 3 to obtain a probabilistic classifier which is then evaluated by means of the area under the ROC curve value. For the two binary-class KRK data set we have thus repeated the rule induction process with only a part of the propositionalized data (750 examples in KRK) and compare RSD rules with the standard covering algorithm.

We have made the comparisons of the performance of the four methods in terms of the area under the ROC curve only in the KRK domain. The reasons for this decision are that

- the Trains domain is too small for splitting the training examples into a training set and a separate test set, and
- the Telecommunications domain is a multi-class problem where the area under ROC curve elevation can not be applied in a straight-forward way.

Table 5 lists the area under ROC values achieved by each of the four combinations of methods on the KRK domain[6]. The values have been obtained by using the subgroup description rules for probabilistic classification as described in Section 3. The corresponding ROC curves are shown in Figure 2. It can be observed the using the WRAcc heuristic improved the predictive performance

[6] Note that we have used a separate test set for evaluating the results - the more elaborate evaluation using cross-validation is left for further work.

Table 5. The area under ROC values for 4 possible combinations of methods on the KRK domain.

KRK	Cov + WRAcc	We + WRAcc	Cov + Acc	We + Acc
Area Under ROC	0.93	0.84	0.92	0.80

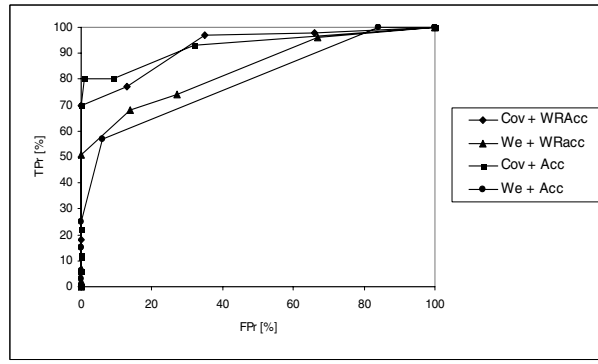


Fig. 2. The ROC curves for 4 possible combinations of methods on the KRK domain.

both with the standard coverage approach and the example weighting approach. We also observe that using example weighting yielded smaller predictive accuracies than standard coverage in both combinations with WRAcc and Acc. Our explanation is that due to the chosen parameterization of slow weight decrease and low total weight threshold we obtained an overly complex predictive model, composed of too many rules, leading to poorer predictive performance. The chosen parameterization was motivated by the aim of obtaining an exhaustive description of possibly interesting subgroups, but alternative, prediction-aimed settings as well as filtering based on significance measures are part of the future experimentation.

5 Conclusions

We have developed the system RSD which discovers interesting subgroups in classified relational data. The strategy followed by the system begins with converting the relational data into a single relation via first-order feature construction. RSD can 'sensitively' extract constants from the input data and employ them in the feature definitions. It can also detect and remove some irrelevant features, but so far only in a relatively simplified way. Finally, to identify interesting subgroups, RSD takes advantage of the method of example-weighting (used in the so-called weighted covering algorithm) and employing the *WRAcc* measure as a heuristic for rule evaluation.

In three experimental domains, we have shown that example weighting improves the average significance of discovered rules with respect to the standard covering algorithm. Also, the use of weighted relative accuracy increases the average coverage (generality) of resulting rules. The combined effect of the two

techniques is thus an important contribution to the problem of subgroup discovery.

From the classification point of view, we observed that the WRAcc heuristic improved the predictive performance on the tested domain compared to the simple accuracy measure heuristic. On the other hand, example-weighting yielded a smaller predictive performance compared to the standard coverage approach. This is in our opinion due to the chosen parameterization of the algorithm motivated by the need of exhaustive description. In future work we will use test the results using cross-validation. We will also experiment with alternative, prediction-aimed settings.

Acknowledgements

The work reported in this paper was supported by the Slovenian Ministry of Education, Science and Sport, the IST-1999-11495 project Data Mining and Decision Support for Business Competitiveness: A European Virtual Enterprise, the British Council project Partnership in Science PSP-18, and the ILPnet2 Network of Excellence in Inductive Logic Programming.

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In U.M. Fayyad, G. Piatetski-Shapiro, P. Smyth and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, 307–328. AAAI Press, 1996.
2. A. Srinivasan and R.D. King. Feature construction with Inductive Logic Programming: A study of quantitative predictions of biological activity aided by structural attributes. *Data Mining and Knowledge Discovery*, 3(1):37–57, 1999.
3. B. Cestnik. Estimating probabilities: A crucial task in machine learning. In L. Aiello, editor, *Proc. of the 9th European Conference on Artificial Intelligence*, 147–149. Pitman, 1990.
4. P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In Y. Kodratoff, editor, *Proc. of the 5th European Working Session on Learning*, 151–163. Springer, 1989.
5. P. Clark and T. Niblett. Induction in noisy domains. In I. Bratko and N. Lavrač, editors, *Progress in Machine Learning (Proc. of the 2nd European Working Session on Learning)*, 11–30. Sigma Press, 1987.
6. P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.
7. S. Džeroski, B. Cestnik, and I. Petrovski. (1993) Using the m-estimate in rule induction. *Journal of Computing and Information Technology*, 1(1):37 – 46, 1993.
8. C. Ferri-Ramírez, P.A. Flach, and J. Hernandez-Orallo. Learning decision trees using the area under the ROC curve. In *Proc. of the 19th International Conference on Machine Learning*, 139–146. Morgan Kaufmann, 2002.
9. P.A. Flach and N. Lachiche. 1BC: A first-order Bayesian classifier. In *Proc. of the 9th International Workshop on Inductive Logic Programming*, 92–103. Springer, 1999.

10. S. Kramer, N. Lavrač and P.A. Flach. Propositionalization approaches to relational data mining. In S. Džeroski and N. Lavrač, editors, *Relational Data Mining*, 262–291. Springer, 2001.
11. M. Kukar, I. Kononenko, C. Grošelj, K. Kralj, and J.J. Fettich. Analysing and improving the diagnosis of ischaemic heart disease with machine learning. *Artificial Intelligence in Medicine*, special issue on *Data Mining Techniques and Applications in Medicine*, 16, 25–50. Elsevier, 1998.
12. N. Lavrač and P. Flach. An extended transformation approach to Inductive Logic Programming. *ACM Transactions on Computational Logic* 2(4): 458–494, 2001.
13. N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
14. N. Lavrač, P. Flach, and B. Zupan. Rule evaluation measures: A unifying view. In *Proc. of the 9th International Workshop on Inductive Logic Programming*, 74–185. Springer, 1999.
15. N. Lavrač, D. Gamberger, and V. Jovanoski. (1999). A study of relevance for learning in deductive databases. *Journal of Logic Programming 40,* 2/3 (August/September), 215–249.
16. R.S. Michalski, I. Mozetič, J. Hong, and N. Lavrač. The multi-purpose incremental learning system AQ15 and its testing application on three medical domains. In *Proc. 5th National Conference on Artificial Intelligence*, 1041–1045. Morgan Kaufmann, 1986.
17. S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4): 245–286, 1995.
18. F. Provost and T. Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42(3), 203–231, 2001.
19. J.R. Quinlan. Learning Logical definitions from relationa. *Machine Learning*, 5(3): 239–266, 1990.
20. R.L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.
21. L. Todorovski, P. Flach, and N. Lavrač. Predictive performance of weighted relative accuracy. In D.A. Zighed, J. Komorowski, and J. Zytkow, editors, *Proc. of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, 255–264. Springer, 2000.
22. S. Wrobel. An algorithm for multi-relational discovery of subgroups. In *Proc. First European Symposium on Principles of Data Mining and Knowledge Discovery*, 78–87. Springer, 1997.
23. F. Železný, J. Zídek, and O. Štěpánková. A learning system for decision support in telecommunications. In *Proc. First International Conference on Computing in an Imperfect World*, 88–101. Springer, 2002.

Mining Frequent Logical Sequences with SPIRIT-LºG

Cyrille Masson[1] and François Jacquenet[2,⋆,⋆⋆]

[1] LISI, INSA Lyon
Bâtiment Blaise Pascal, 69621 Villeurbanne, France
Cyrille.Masson@lisi.insa-lyon.fr
[2] EURISE, Université de Saint-Etienne
23 rue du Docteur Paul Michelon, 42023 Saint-Etienne Cedex 2, France
Francois.Jacquenet@univ-st-etienne.fr

Abstract. Sequence mining is an active research field of data mining because algorithms designed in that domain lead to various valuable applications. To increase efficiency of basic sequence mining algorithms, generally based on a levelwise approach, more recent algorithms try to introduce some constraints to prune the search space during the discovery process. Nevertheless, existing algorithms are actually limited to extract frequent sequences made up of items of a database. In this paper, we generalize the notion of sequence to define what we call logical sequence where each element of a sequence may contain some logical variables. Then we show how we can extend constrained sequence mining to constrained frequent logical sequence mining[1].

Keywords: Data Mining, Sequence Mining, First Order Logic.

1 Introduction

Sequence mining in databases is an active field of data mining. Indeed, it leads to various valuable applications. For instance, we may cite the discovery of frequent sequences of purchases in department stores [2] or of UNIX commands [8]. Classical algorithms for the discovery of frequent sequences [2,10] rely on levelwise methods designed for association rule mining [1]. Nevertheless, these algorithms suffer from a lack of interactivity with the user. Indeed, selective users having an idea of some properties of the searched patterns can be quickly overwhelmed with many irrelevant results. Some works have been done to address this problem by pushing constraints into the sequence mining process [13,14,7].

Moreover, such algorithms discover sequences in propositional logic, whereas many researches have shown the interest of discovering knowledge expressed in first order logic. The usefulness of Inductive Logic Programming in multi-relational data mining is now widely recognized. WARMR [4] has thus been

⋆ Research partially funded by the European contract cInQ IST 2000-26469-FET.
⋆⋆ Paper presented with the financial help of ILPnet2.
[1] This paper is an extended english version of [11].

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 166–182, 2003.

designed to extend association rule mining to frequent queries mining in a deductive database and TILDE [3] extends decision tree building to logical decision tree building. The reader interested in an good overview of the ILP techniques for relational data mining is invited to refer to [5].

In this paper, we are interested in the discovery of logical sequences, i.e. sequences built with atoms of first order logic. More particularly, we want to mine frequent logical sequences, i.e. that occur "often enough" in the input dataset. The following example shows the interest of such an approach. Let us consider the discovery of shell scripts from logs of an Unix system [8] and suppose that we have a database of commands entered by some users, and which has been translated into literals (cf. Table 1). With an user-given minimal frequency threshold of 30 %, algorithms presented in this paper are able to extract from this dataset frequent logical sequences of the form: $\langle mv(F, D)\ cd(D)\ emacs(F)\ gcc(F, E)\ exec(E)\ \rangle$, where F, D, E are logical variables. Indeed, this sequence occurs in two of the five input sequences. It is easy to understand the usefulness of such sequences in the design of an "intelligent" shell which can guide the user in a series of commands he has to submit to the operating system.

However, as mining all frequent logical sequences will overwhelm the user with many results, we choose here to also consider a regular expression constraint, like the one already proposed by Garofalakis in SPIRIT algorithms [7]. It aims at constraining the syntactical aspect of extracted sequences. So, we consider here the problem of adapting the SPIRIT algorithms to the problem of logical sequences extraction.

In Section 2 of this paper, we give an overview of the SPIRIT principles and of the work already done in the field of temporal pattern mining in first order logic. Then, in Section 3, we formally define logical sequences and the constraints we are going to consider. In Section 4, we show how we have adapted the SPIRIT family of algorithms to the discovery of logical sequences in order to obtain a new family of algorithms called SPIRIT-LºG. In section 5, we run some experiments to show the efficiency of the various methods of this family. In section 6, we discuss results of our work and give some perspectives.

Table 1. Sequences of literals

User 1	User 2	User 3	User 4	User 5
mv(ex.c,src/)	mv(f1,f2)	exec(mail)	mv(mn.c,prj/)	cd(docs)
exec(mail)	cd(/tmp)	ftp(srv1)	cd(prj/)	gv(f1.ps)
cd(src/)	rm(train.dat)	exec(xcdroast)	exec(lpq)	gv(f2.ps)
emacs(ex.c)	cd(∼)	exec(logout)	emacs(mn.c)	lpr(f2.ps)
talk(user2)	emacs(n.tex)		gcc(mn.c,prj)	cd(../progs)
gcc(ex.c,ex)	latex(n)		exec(prj)	exec(make)
exec(ex)	dvips(n.tex,n.dvi)		exec(logout)	exec(proto)
gprof(ex)	gv(n.ps)			more(res)
exec(lpq)	lpr(n.ps)			exec(logout)
exec(logout)	exec(logout)			

2 Related Work

In the area of sequences of items, a lot of work has been done to push constraints inside the mining process. The GSP algorithm [13] aims at increasing the efficiency of algorithms of [2] by using some constraints such as taxonomies or a temporal window. The cSPADE algorithm [14],proposed by Zaki, incorporates some new constraints in the mining process, like the restriction of the length and the width of the sequences, or the definition of a minimum and/or a maximum gap between two consecutive elements of a sequence, and is able to work either in a breadth-first way or a depth-first way.

But, the approach which is the basis of our work is the family of SPIRIT algorithms [7,6], designed by Minos N. Garofalakis. It considers the mining of sequences of items and proposed to use the language of regular expressions built on the alphabet of items to specify a syntactic constraint over sequences to be discovered. That leads to a constraint $\mathcal{C}_{\mathcal{A}}$ restricting the items and the order of the items occuring in mined sequences. If we denote C_{Freq} the usual frequency constraint, then SPIRIT is a solver for mining sequences satisfying $\mathcal{C}_{\mathcal{A}} \wedge \mathcal{C}_{Freq}$. As $\mathcal{C}_{\mathcal{A}}$ is generally difficult to actively use in the extraction process, Garofalakis proposed to consider different relaxations of $\mathcal{C}_{\mathcal{A}}$ and for each of them, he designed new methods for candidate generation and pruning. Then, a post-processing step filters out sequences not satisfying $\mathcal{C}_{\mathcal{A}} \wedge \mathcal{C}_{Freq}$. With these different algorithms, he observed a trade-off between the frequency based pruning and the pruning due to the use of the regular expression constraint.

In the domain of extracting temporal sequences made of atoms, few papers have been written. However, we can cite two major approaches:

- Using WARMR to extract temporal information [8]. In this approach, temporal information is explicitely encoded in the atoms and the extraction is performed using the language bias offered by WARMR to incorporate background knowledge on the atoms order. However, the "direct" use of WARMR can be less efficient that using an hybrid method involving both a minimal occurence algorithm for the propositional task (detecting frequent subsequences of commands) and WARMR for the relational task (detecting frequent patterns in parameters of commands). The main advantage of using WARMR is that it allows to express constraints on the parameters of the atoms, what is particuliarly useful for reducing the search space.
- MineSeqLog [9], which is an inductive database mining system for mining and querying sequential data expressed in first order logic. MineSeqLog efficiency processes inductive queries on sequences, by using some concepts used with version-spaces. Moreover, it uses background knowledge and takes into account a conjunction of monotonic and anti-monotonic constraints.

These two approaches for the mining of sequences of atoms cannot perfectly fulfil our requirements, because we want to incorporate a syntactical constraint on sequences to be mined, and this constraint is not necessarily either monotonic or anti-monotonic. That is why we choose to adapt SPIRIT algorithms to the discovery of logical sequences made of atoms.

3 Logical Sequence Mining

We consider here the classical framework of first order logic. So, we consider a set $\mathcal{V}ar$ of variables and a set $\mathcal{C}st$ of constants such that $\mathcal{V}ar \cap \mathcal{C}st = \emptyset$. Variables may be distinguished from constants by the fact that their characters are capitalized. $\mathcal{P}red$ is a set of predicate symbols and we suppose that we have the usual definitions of substitution and unification. Finally, if $p \in \mathcal{P}red$ has an arity of n and $\forall i, 1 \leq i \leq n, t_i \in \mathcal{V}ar$ then $p(t_1, t_2, \ldots, t_n)$ is called a *free atom*.

We consider, as an input of our system, a database $\mathcal{D}$ of *sequences*, where each sequence is an ordered list of ground atoms. These sequences are called *input sequences*. Notice that, for our first implementation , we have choosen to restrict ourselves to the case of input sequences corresponding to list of ground atoms, as we do not have designed yet algorithms for the extraction in input sequences made of lists of sets of atoms. In the remainder of this paper, the list of the elements of a sequence s will be denoted $\langle s_1\ s_2 \ldots s_n \rangle$ where s_i is the i^{th} element of s. The length $|s|$ of a sequence is equal to the number of elements in the sequence. A sequence of length k is called a k-sequence.

The language of patterns we are going to consider is the one of logical sequences. A *logical sequence* is an ordered list of free atoms. We note $V(s)$ the set of variables that appear in a logical sequence s. The concatenation of two logical sequences s and t is noted $\langle s\ t \rangle$. If we consider two logical sequences $s = \langle s_1 \ldots s_n \rangle$ and $t = \langle t_1 \ldots t_m \rangle$, we say that s is a *logical subsequence* of t if there exists integers $j_1 < j_2 < \ldots < j_n$ such that $s = \langle t_{j_1}\ \ldots\ t_{j_n} \rangle$. Let $s = \langle s_1 \ldots\ s_n \rangle$ be a logical sequence, and θ be a substitution, we denote θs the application of the substitution θ to the logical sequence s and we have $\theta s = \langle \theta s_1\ \ldots \theta s_n \rangle$. A logical sequence s is more general than a sequence t if there exists a substitution θ s.t. $\theta s = t$. For instance, let $s = \langle p_4(X, Y)\ p_3(Z, T) \rangle$ and $t = \langle p_4(U, V)\ p_3(U, R) \rangle$. Then s is more general than t (using the substitution $\theta = \{X \mid U, Y \mid V, Z \mid U, T \mid R\}$).

We now define the containment relationship between input sequences and logical sequences. We say that an input sequence t *contains* a logical sequence s if there exists a substitution θ and integers $j_1 < j_2 < \ldots < j_n$ such that $\theta s_1 = t_{j_1}$, ..., $\theta s_n = t_{j_n}$. We define the *support* of a logical sequence s as the proportion of input sequences of $\mathcal{D}$ that contain s. Given a set of logical sequences $\mathcal{S}$, we say that $s \in \mathcal{S}$ is *maximal* if there exists no sequence $t \in \mathcal{S} - \{s\}$ such that s is a logical subsequence of t. Given an user-defined threshold σ, a logical sequence is *frequent* if its support is greater than σ. We can then define a constraint $\mathcal{C}_{Freq}$. To count the support of candidate logical sequences and to know which logical sequences are included in a given input sequences, we have adapted the hash-tree structure described in [13]. When scanning the database, for each input sequence, we look into the hash-tree storing all candidates to find the set of logical sequences that can be contained in the input sequence. Finally, we check if there exists a substitution that makes true the containement relationship between the input sequence and the candidate logical sequence. In the worst of cases, the search space is exponential in the number of ground atoms of the input sequence. However, in practice, the choice of the hash-tree structure allows to significantly speed up the search [13].

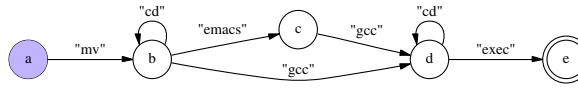


Fig. 1. DFA associated to the Regular Expression $mv\ cd^*\ (emacs\ gcc|gcc)\ cd^*\ exec$

As we have decided to adapt the regular expression constraint proposed by Minos N. Garofalakis in SPIRIT to the case of logical sequences, we consider an user-defined regular expression $\mathcal{R}$ built on the alphabet of predicate symbols $\mathcal{P}red$. We will denote $\mathcal{A}$ the Deterministic Finite Automaton associated to $\mathcal{R}$. Notice that the fact that $\mathcal{A}$ is deterministic is essential, as algorithms proposed further in this paper can only work with this kind of automata. Here, we say that a logical sequence s is accepted by $\mathcal{A}$ if the ordered list of predicate symbols of s corresponds to a string accepted by $\mathcal{A}$. We can now define a constraint $\mathcal{C}_\mathcal{A}$: a logical sequence s verifies $\mathcal{C}_\mathcal{A}$ iff s is accepted by $\mathcal{A}$. For instance, if we consider the regular expression $\mathcal{R} = mv\ cd^*\ (emacs\ gcc|gcc)\ cd^*\ exec$ (which associated DFA $\mathcal{A}$ is depicted on Figure 1), we can say that the logical sequence $\langle mv(F,D)\ cd(D)\ emacs(F)\ gcc(F,E)\ exec(E)\rangle$ verifies $\mathcal{C}_\mathcal{A}$.

A constraint $\mathcal{C}$ is *anti-monotonic*, if for a logical sequence s verifying $\mathcal{C}$, all its logical subsequences also verify $\mathcal{C}$. This property is very useful to push constraints inside levelwise algorithms because it ensures a safe pruning of the search space [12]. Indeed, if a sequence s does not satisfy an anti-monotonic constraint, no sequence in which s is included will verify this constraint.

Definition 1 *A logical sequence s is legal with respect to a state b of the automaton $\mathcal{A}$ if each transition of $\mathcal{A}$ is defined from b following the series of predicate symbols on which the atoms of s are built.*

Example: If we consider the automaton of Figure 1,we can say that the logical sequence $\langle emacs(X)\ gcc(X,Y)\rangle$ is legal w.r.t. state b.

Definition 2 *A logical sequence s is valid with respect to a state b of $\mathcal{A}$ if it is legal with respect to b, and if the final state of the series of transitions on the predicate symbols of s from b is a final state of the automaton.*

Definition 3 *We say that a logical sequence s is valid if it s is valid w.r.t. the initial state of the automaton $\mathcal{A}$.*

Example: With the automaton of Figure 1, $\langle gcc(X,Y)\ exec(Y)\rangle$ is valid w.r.t. the state b and $\langle mv(X,Y)\ \ gcc(Y,Z)\ \ exec(Z)\rangle$ is valid.

Given all these definitions, we can now define the problem of mining frequent logical sequences under a regular expression constraint as follows:

- **Given** $\mathcal{D}$, a database of input sequences, a minimum threshold specified by the user, and a constraint given as a regular expression $\mathcal{R}$ (or as the associated Deterministic Finite Automaton $\mathcal{A}$).
- **Find** all the logical sequences that are valid and frequent in $\mathcal{D}$.

4 SPIRIT-LºG for Mining Frequent Logical Sequences

4.1 Generic Algorithm SPIRIT-LºG

Algorithm 1 gives the general structure of the mining process for logical sequences with a constraint on the syntactic structure of the sequences specified as a regular expression. This algorithm is similar to the one given in [7] that aims at mining sequences of items. The algorithm consists in a levelwise search that extracts at each step, some longer and longer logical sequences. C_k is the set of candidate logical k-sequences and F_k is the set of frequent logical k-sequences.

Algorithm 1: Generic algorithm SPIRIT-LºG for mining frequent logical sequences

Input: $\mathcal{D}$ a database of input sequences, $\mathcal{C}$ a constraint specified as a regular expression, a threshold *minsup*

Output: set F of all frequent logical sequences satisfying $\mathcal{C}$

$\mathcal{C}'$ = a relaxed constraint of $\mathcal{C}$;
$F = F_1 = \{$ frequent logical 1-sequences of $\mathcal{D}$ that satisfy $\mathcal{C}'$ $\}$;
// Pruning of F_1 by specialization
$F_1 = \{s \in F_1, \exists t \in F_1, t$ is more general than s and t is contained in the same input sequences as $s\}$;
$k \leftarrow 2$;
repeat
  // Generation of candidates
  Use $\mathcal{C}'$ and F to generate $C_k :=$ {candidate logical k-sequences satisfying $\mathcal{C}'$};
  // Pruning of candidates
  $P = \{s \in C_k, \exists t$ logical subsequence of s that satisfy $\mathcal{C}'$ and that do not unify with any logical sequence of $F\}$;
  $C_k = C_k - P$;
  // Counting of candidates
  Traverse $\mathcal{D}$ to count the candidate k-sequences of atoms of C_k;
  $F_k =$ frequent logical sequences of C_k;
  // Pruning of F_k by specialization
  $F_k = \{s \in F_k, \exists t \in F_k, t$ is more general than s and t is contained in the same input sequences as $s\}$;
  $F = F \cup F_k$;
  $k \leftarrow k + 1$;
until StopCondition(F, $\mathcal{C}'$);
// Post processing of F : we strengthen the constraint by using $\mathcal{C}$
Return the logical sequences of F that satisfy $\mathcal{C}$;

As the regular expression constraint $\mathcal{C}$ is generally not anti-monotonic, it is hard to actively use it with the classical levelwise paradigm. That is why

Garofalakis proposed to consider relaxations $\mathcal{C}'$ of $\mathcal{C}$ [6] for which it is possible to design *ad hoc* generating and pruning functions allowing an effective use of $\mathcal{C}$. Thus, we have an algorithm returning more results than originally expected as it works with a conjunction of constraint $\mathcal{C}' \wedge \mathcal{C}_{Freq}$. Results satisfying $\mathcal{C} \wedge \mathcal{C}_{Freq}$ are among those results. So, a post processing step is needed to only keep relevant results. We now present how we have adapted the four versions of SPIRIT into SPIRIT-LºG algorithms to extract frequent logical sequences.

4.2 SPIRIT-LºG(N)

SPIRIT-LºG(N) considers as a constraint $\mathcal{C}'$ the fact all the predicate symbols on which the atoms of a candidate logical sequence of C_k are built must appear in the regular expression $\mathcal{R}$. This constraint is trivially anti-monotonic, so generation and pruning of candidates can be done in a similar way to classical Apriori-like algorithms for mining sequences of items.

Generation of Candidates. We generate new candidates for all pair (s, t) of sequences such that the suffix of length $(k-2)$ of s unifies with the prefix of length $(k-2)$ of t. More formally, let $s = \langle s_1 \ \ldots s_{k-1} \rangle$ and $t = \langle t_1 \ \ldots t_{k-1} \rangle$ two logical sequences, if needed we rename the variables of s and t in such a way that $V(s) \cap V(t) = \emptyset$. If there exists θ such that $t_j = \theta s_{j+1}$ for $1 \leq j \leq k-2$, then we add to C_k all the candidates of the form $\langle \theta s \ \theta' t_{k-1} \rangle$ where θ' is a substitution of a part of $V(t_{k-1})$ with some variables of $V(\theta s_1)$. To find all the possible substitutions θ', we call the procedure Enum-Subst$(V(t_{k-1}), V(\theta s_1))$.

We may note that, for two given sets of variables V and W, the total number of substitutions generated by the algorithm Enum-Subst is: $\sum_{k=0}^{|V|} C_{|V|}^{k} |W|^k$, i.e. $(1 + |W|)^{|V|}$ This result means that, in the worst case, the number of generated candidates can be exponential in the arity of the last predicate of the candidate sequences. That is why, in our experiments (Section 5), we have worked with predicate symbols of small arity.

Example. Suppose we have discovered the two logical sequences $s = \langle p(X, R) \ q(Y) \ \ r(X, S) \rangle$ and $t = \langle q(L) \ r(N, T) \ s(J, T) \rangle$. Considering the substitution $\theta = \{Y \mid L, X \mid N, S \mid T\}$, we call the procedure Enum-Subst$(\{J, T\}, \{X, R\})$

Algorithm 2: Enum-Subst

Input: V, W two sets of variables
$\Omega = \{\epsilon$ the identity substitution$\}$;
foreach *part* $P \in \mathcal{P}(V)$ **do**
  $\Omega = \Omega \cup \{w, w$ is a substitution of all the variables of P with some variables of $W\}$;
end
return Ω

which returns the following possible set of substituions to consider:

$$\Omega = \{\epsilon, \{J \mid X\}, \{T \mid X\}, \{J \mid R\}, \{T \mid R\}, \{J \mid X, T \mid X\} \\ \{J \mid R, T \mid R\}, \{J \mid X, T \mid R\}, \{J \mid R, T \mid X\}\}$$

Thus, we add 9 candidates to the set of candidate logical sequences.

Pruning of Candidates. A candidate logical sequence s is pruned from C_k if there exists a logical $(k-1)$-subsequence of s that does not unify with any logical sequence of F_{k-1}.

Stop Condition. The set of logical k-sequences F_k is empty. Then we cannot generate any candidate at step $(k+1)$.

4.3 SPIRIT-LºG(L)

In SPIRIT-LºG(L), $\mathcal{C}'$ specifies that each candidate logical sequence must be legal w.r.t. a state of the automaton $\mathcal{A}$. The algorithm uses $\mathcal{A}$ to perform the pruning. $F_k(b)$ denotes the set of logical k-sequences legal w.r.t. a state b of $\mathcal{A}$.

Generation of Candidates. For each state b of $\mathcal{A}$, we add to C_k the candidate k-sequences that are legal with respect to b and potentially frequent.

Lemma 1 *Let $s = \langle s_1 \dots s_k \rangle$ be a candidate logical sequence legal w.r.t. a state b of $\mathcal{A}$, where $b \xrightarrow{p} c$ is a transition of $\mathcal{A}$ and where p is the predicate symbol on which s_1 is built. For s to be frequent, $\langle s_1 \dots s_{k-1} \rangle$ must unify with a logical sequence of $F_{k-1}(b)$ and $\langle s_2 \dots s_k \rangle$ must unify with a logical sequence of $F_{k-1}(c)$.*

The candidate logical sequences for a state b will be built as follows. For all sequence s in $F_{k-1}(b)$, if $b \xrightarrow{p} c$ is a transition of $\mathcal{A}$ and s_1 is built on the predicate symbol p, then for all sequence t in $F_{k-1}(c)$ such that:

- $V(s) \cap V(t) = \emptyset$ (we can rename variables of s and t so that it is the case).
- $\langle s_2 \dots s_{k-1} \rangle$ and $\langle t_1 \dots t_{k-2} \rangle$ may be unified (we denote θ the substitution such that $t_j = \theta s_{j+1}$, for $1 \leq j \leq k-2$).

we add to C_k all the logical sequences of the form $\langle \theta s \ \theta' t_{k-1} \rangle$, where θ' is a substitution of a part of $V(t_{k-1})$ with some variables of $V(\theta s_1)$. To find all the possible substitutions, we call the procedure Enum-Subst($V(t_{k-1})$, $V(\theta s_1)$).

Pruning of Candidates. To prune a candidate s, we must find a logical subsequence of s which is legal w.r.t. a state of $\mathcal{A}$ and that does not unify with any frequent sequence already discovered. So, we calculate the maximal subsequences of s of length smaller than k, that are legal w.r.t. a state of $\mathcal{A}$. If one of them does not unify with any logical sequence of F, then s is pruned from C_k.

We need to find all the maximal legal logical subsequences of a candidate sequence s. For that, we can use Algorithm 3 which is the straightforward adaptation of the algorithm presented in [6]. Given an automaton $\mathcal{A}$ which sets of

Algorithm 3: FindMaxSubseqLog to find maximal subsequences of atoms

Input: $\mathcal{A}$ an automaton, S a set of initial states, E a set of final states, s a sequence of atoms

foreach state b of $\mathcal{A}$ **do** maxSeq[b] $\leftarrow \emptyset$;
for k from $|s|$ to 1 **do**
  foreach state b of $\mathcal{A}$ **do**
    tmpSeq[b] $\leftarrow \emptyset$;
    if $\exists$ a transition $b \xrightarrow{p} c$ in $\mathcal{A}$ and that s_k is built on the symbol p **then**
      if $c \in E$ **then** tmpSeq[b] $\leftarrow \{s_k\}$
      tmpSeq[b] $\leftarrow$ tmpSeq[b] $\cup \{\langle s_k\ t\rangle : t \in$ maxSeq[c]$\}$;
  end
  foreach state b of the automaton $\mathcal{A}$ **do**
    maxSeq[b] $\leftarrow$ maxSeq[b] $\cup$ tmpSeq[b];
    foreach sequence t in MaxSeq[b] **do**
      if $\exists\ u \in$ maxSeq[b]-$\{\langle s_k \dots s_{|s|}\rangle\}$ such that t is a subsequence of u **then** Prune t from maxSeq[b];
    end
  end
end
return $\bigcup_{b \in S}$ maxSeq[b]-$\{s\}$ (after the pruning of non maximal sequences)

initial and final states (S and E) are given as parameters, it returns all the valid logical subsequences of s by actually first computing all the logical subsequences of s that are valid w.r.t. a state of $\mathcal{A}$ and only returning those that are valid w.r.t. a state in S. Proofs of soundness and completeness are directly adapted from [6], (except that we are reasoning on predicate symbols instead of items) and mainly rely on the following fact. Let maxSeq(b, s) be the set of maximal subsequences of $s = \langle l_1 \dots l_{|s|}\rangle$ that are valid w.r.t. a state b and $t = \langle l_2 \dots l_{|s|}\rangle$, a superset of maxSeq$(b, s)$ can be computed from maxSeq(b, t) using the fact that:

- $maxSeq(b, s) \subseteq maxSeq(b, t) \cup \{\langle s_1\ u\rangle$ with $u \in maxSeq(c, t)\} \cup \{s_1\}$ where s_1 is an atom of s built $p \in \mathcal{P}red$, and where $b \xrightarrow{p} c$ is a transition of $\mathcal{A}$.
- $maxSeq(b, s) \subseteq maxSeq(b, t)$ elsewhere.

A post-processing step is then needed to only keep maximal subsequences.
Thus, to find all maximal legal logical subsequences of a candidate logical sequence s, we call FindMaxSubseqLog with both S and E sets equals to the set of all states of $\mathcal{A}$.

Stop Condition. The set of frequent logical k-sequences that are legal w.r.t. the initial state a of $\mathcal{A}$ is empty, i.e. $F_k(a)$ is empty. We will not be able to generate any candidate logical $(k+1)$-sequences which is legal w.r.t. a.

4.4 SPIRIT-LºG(V)

SPIRIT-LºG(V) uses a stronger constraint $\mathcal{C}'$ than in SPIRIT-LºG(L) as we impose that all generated candidates are valid w.r.t. a state b of the automaton $\mathcal{A}$. $F_k(b)$ denotes the set of frequent k-sequences that are valid w.r.t. b.

Generation of Candidates. If a candidate logical sequence $s \in C_k$ is valid w.r.t. a state b of $\mathcal{A}$, and $b \xrightarrow{p} c$ is a transition of $\mathcal{A}$ and s_1 is a free atom built on $p \in \mathcal{P}red$, then the logical subsequence $\langle s_2 \dots s_k \rangle$ which is the suffix of length $(k-1)$ of s unifies with a frequent logical sequence of F which is valid w.r.t. c.

This property gives us an interesting process to generate candidates. For all transition $b \xrightarrow{p} c$ where p is a predicate symbol of arity n, for all logical sequence $t \in F_{k-1}(c)$, such that $V(t) \cap \{X_1, \dots, X_n\} = \emptyset$ (even by renaming, if necessary, the variables of t), we add to the set of candidates for state b the k-sequences of the form $\langle \theta p(X_1, \dots, X_n)\ t \rangle$ where θ is a substitution of a part of $\{X_1, \dots, X_n\}$ with some variables of $V(t)$. To find the set of these substitutions, we call the procedure Enum-Subst($\{X_1, \dots, X_n\}, V(t)$). The union of all these sets of candidates for all the states of $\mathcal{A}$ gives C_k.

Example. Suppose we have discovered the logical sequence $s = \langle q(X,T)\ r(T,K) \rangle$ in F_2, and that it is valid w.r.t. a state c. If there is a transition $b \xrightarrow{p} c$ and the arity of p is 2, calling Enum-Subst($\{X_1, X_2\}$, $\{X, T, K\}$) returns:

$$\Omega = \{\epsilon, \{X_1/X\}, \{X_1/T\}, \{X_1/K\}, \{X_2/X\}, \{X_2/T\}, \{X_2/K\}, \{X_1/X, X_2/X\}, \{X_1/X, X_2/T\}, \{X_1/X, X_2/K\}, \{X_1/T, X_2/X\}, \{X_1/T, X_2/T\}, \{X_1/T, X_2/K\}, \{X_1/K, X_2/X\}, \{X_1/K, X_2/T\}, \{X_1/K, X_2/K\}\}$$

Then we add to C_3 the following candidates:

- With $\theta = \epsilon$, we add $\langle p(X_1, X_2)\ q(X,T)\ r(T,K) \rangle$
- With $\theta = \{X_1/X\}$, we add $\langle p(X, X_2)\ q(X,T)\ r(T,K) \rangle$
- With $\theta = \{X_1/T\}$, we add $\langle p(T, X_2)\ q(X,T)\ r(T,K) \rangle$
- With $\theta = \{X_1/K\}$, we add $\langle p(K, X_2)\ q(X,T)\ r(T,K) \rangle$
- With $\theta = \{X_2/X\}$, we add $\langle p(X_1, X)\ q(X,T)\ r(T,K) \rangle$
- With $\theta = \{X_2/T\}$, we add $\langle p(X_1, T)\ q(X,T)\ r(T,K) \rangle$
- With $\theta = \{X_2/K\}$, we add $\langle p(X_1, K)\ q(X,T)\ r(T,K) \rangle$
- With $\theta = \{X_1/X, X_2/X\}$, we add $\langle p(X, X)\ q(X,T)\ r(T,K) \rangle$
- With $\theta = \{X_1/X, X_2/T\}$, we add $\langle p(X, T)\ q(X,T)\ r(T,K) \rangle$
- With $\theta = \{X_1/X, X_2/K\}$, we add $\langle p(X, K)\ q(X,T)\ r(T,K) \rangle$
- With $\theta = \{X_1/T, X_2/X\}$, we add $\langle p(T, X)\ q(X,T)\ r(T,K) \rangle$
- With $\theta = \{X_1/T, X_2/T\}$, we add $\langle p(T, T)\ q(X,T)\ r(T,K) \rangle$
- With $\theta = \{X_1/T, X_2/K\}$, we add $\langle p(T, K)\ q(X,T)\ r(T,K) \rangle$
- With $\theta = \{X_1/K, X_2/X\}$, we add $\langle p(K, X)\ q(X,T)\ r(T,K) \rangle$
- With $\theta = \{X_1/K, X_2/T\}$, we add $\langle p(K, T)\ q(X,T)\ r(T,K) \rangle$
- With $\theta = \{X_1/K, X_2/K\}$, we add $\langle p(K, K)\ q(X,T)\ r(T,K) \rangle$

Algorithm 4: GenR

Input: s a logical sequence, b the current state, B the set of traversed states

foreach *transition* $b \xrightarrow{p} c$ *in* $\mathcal{A}$ **do**
  Let $l = p(X_1 \dots X_n)$ an atom built on p;
  Rename if necessary variables of s s.t. $V(s) \cap \{X_1, \dots, X_n\} = \emptyset$
  if *(*$|s| = k-1$ **and** c is a final state*)* **then**
    $\Omega = Enum\text{-}Subst(\{X_1, \dots, X_n\}, V(s))$
    $C_k = C_k \cup \{\langle s\ \theta' l\rangle, \theta' \in \Omega\}$;
  if *(*$|s| \neq k-1$ **and** *(*c is not a final state **or** *(*$\exists l \in F$ *and a unifier* $\theta, l = \langle \theta s\ l_{|s|+1}\rangle$ *where* $l_{|s|+1}$ *is built on the predicate symbol* p *))* **then**
    if $c \in B$ **then**
      Let $s = \langle t\ u\rangle$, where t is the prefix of s for which $a \overset{t}{\Rightarrow} c$;
      $T = \{(t', v', \theta) : \langle t'\ v'\rangle \in F_{k-|u|-1}, \exists$ a unifier $\theta, t = \theta t'\}$;
      foreach $(t', v', \theta) \in T$ **do**
        $\Omega' = Enum\text{-}Subst(V(\theta v'), V(u) \backslash V(t))$;
        foreach $\theta' \in \Omega'$ **do**
          $\Omega'' = Enum\text{-}Subst(\{X_1, .., X_n\}, V(s) \cup V(\theta' \theta v'))$;
          $C_k = C_k \cup \{\langle s\ \theta'' l\ \theta' \theta v\rangle, \theta'' \in \Omega''\}$;
        end
      end
    else
      $\Omega = Enum\text{-}Subst(\{X_1, \dots, X_n\}, V(s))$
      foreach $\theta' \in \Omega$ **do** GenR$(\langle s\ \theta' l\rangle, c, B \cup \{b\})$
    end
end

Pruning Candidates. This step is similar to the one of SPIRIT-L^{o}G(L). The idea is to prune from C_k every candidate s having a subsequence which is valid w.r.t. a state of $\mathcal{A}$ and that does not unify with any frequent logical sequence of F. Thus, given a logical sequence, we need to calculate its subsequences that are valid w.r.t. a state of $\mathcal{A}$. For that, we call FindMaxSubseqLog with S equal to the set of all states of $\mathcal{A}$ and E containing the final states of $\mathcal{A}$.

Stop Condition. The set F_k of frequent logical k-sequences valid w.r.t. a state of $\mathcal{A}$ is empty. We thus will be unable to generate candidates at the next step.

4.5 SPIRIT-L^{o}G(R)

In this variant, we do not relax the constraint specified using $\mathcal{R}$. So, we require that each generated candidate sequence is valid, i.e. $\mathcal{C}' = \mathcal{C}$ in Algorithm 1.

Generation of Candidates. To generate candidates in C_k, we have to enumerate all the candidate logical sequences of size k. Algorithm 4 is an adaptation of the procedure proposed in [6] to generate the candidates. We traverse all the transitions of $\mathcal{A}$ and we enumerate all the logical sequences that can be built on the series of predicate symbols making up a path from an initial state to a final state of $\mathcal{A}$. To optimize the candidate generation, we use two optimizations:

- The first one uses the fact that if a logical sequence s is valid but not frequent, then it is not useful to extend it because it cannot lead to a frequent sequence.
- The second one uses the existence of loops in $\mathcal{A}$. For example, suppose that for a logical sequence $\langle t\ u\rangle$ (of length smaller than or equal to k), t and $\langle t\ u\rangle$ lead to the same state from the initial state a. Then, if the sequence $\langle t\ u\ v\rangle$ built by extending $\langle t\ u\rangle$ with a logical sequence v leads to a candidate k-sequence, then $\langle t\ v\rangle$ is also frequent and valid.

Example Consider the automaton of Figure 2 and suppose we have already built, in F_3 the sequence $\langle p(A,B)\ q(C,A)\ t(A,D)\rangle$. While searching the candidate 5-sequences, we build among others a candidate logical sequence $\langle p(Y_1,Y_2)\ q(Y_3,Y_1)\ \ r(Y_1,Y_4)\rangle$. At that time we consider the transition $3 \xrightarrow{s} 2$, where the arity of s is equal to 1. As state 2 has already been traversed, we have $t = \langle p(Y_1,Y_2)\ q(Y_3,Y_1)\rangle$ and $u = \langle r(Y_1,Y_4)\rangle$. In the set T of algorithm 3, we get the triple (t', v', θ) with $t' = \langle p(A,B)\ q(C,A)\rangle$, $v' = \langle t(A,D)\rangle$ and $\theta = \{A/Y_1, B/Y_2, C/Y_3\}$. To calculate Ω', we call Enum-Subst($\{D\}, \{Y_4\}$), and we get $\Omega' = \{\epsilon,\ \{D/Y_4\}\}$. Then we consider each substitution θ' of Ω':

- For $\theta' = \epsilon$, we calculate Ω'' calling Enum-Subst($\{X_1\}, \{Y_1, Y_2, Y_3, Y_4, D\}$). Then we add the following candidates:
  - With $\theta'' = \epsilon$, we add $\langle p(Y_1,Y_2)\ q(Y_3,Y_1)\ r(Y_3,Y_4)\ s(X_1)\ t(Y_1,D)\rangle$
  - With $\theta'' = \{X_1/Y_1\}$, we add $\langle p(Y_1,Y_2)\ q(Y_3,Y_1)\ r(Y_3,Y_4)\ s(Y_1)\ t(Y_1,D)\rangle$
  - With $\theta'' = \{X_1/Y_2\}$, we add $\langle p(Y_1,Y_2)\ q(Y_3,Y_1)\ r(Y_3,Y_4)\ s(Y_2)\ t(Y_1,D)\rangle$
  - With $\theta'' = \{X_1/Y_3\}$, we add $\langle p(Y_1,Y_2)\ q(Y_3,Y_1)\ r(Y_3,Y_4)\ s(Y_3)\ t(Y_1,D)\rangle$
  - With $\theta'' = \{X_1/Y_4\}$, we add $\langle p(Y_1,Y_2)\ q(Y_3,Y_1)\ r(Y_3,Y_4)\ s(Y_4)\ t(Y_1,D)\rangle$
  - With $\theta'' = \{X_1/D\}$, we add $\langle p(Y_1,Y_2)\ q(Y_3,Y_1)\ r(Y_3,Y_4)\ s(D)\ t(Y_1,D)\rangle$
- For $\theta' = \{D/Y_4\}$, we calculate Ω'' calling Enum-Subst($\{X_1\}, \{Y_1, Y_2, Y_3, Y_4\}$). Then we add the following candidates:
  - With $\theta'' = \epsilon$, we add $\langle p(Y_1,Y_2)\ q(Y_3,Y_1)\ r(Y_3,Y_4)\ s(X_1)\ t(Y_1,Y_4)\rangle$
  - With $\theta'' = \{X_1/Y_1\}$, we add $\langle p(Y_1,Y_2)\ q(Y_3,Y_1)\ r(Y_3,Y_4)\ s(Y_1)\ t(Y_1,Y_4)\rangle$
  - With $\theta'' = \{X_1/Y_2\}$, we add $\langle p(Y_1,Y_2)\ q(Y_3,Y_1)\ r(Y_3,Y_4)\ s(Y_2)\ t(Y_1,Y_4)\rangle$
  - With $\theta'' = \{X_1/Y_3\}$, we add $\langle p(Y_1,Y_2)\ q(Y_3,Y_1)\ r(Y_3,Y_4)\ s(Y_3)\ t(Y_1,Y_4)\rangle$
  - With $\theta'' = \{X_1/Y_4\}$, we add $\langle p(Y_1,Y_2)\ q(Y_3,Y_1)\ r(Y_3,Y_4)\ s(Y_4)\ t(Y_1,Y_4)\rangle$

Pruning of Candidates. We prune a candidate logical sequence $s \in C_k$ if it contains a valid logical subsequence that does not unify with any logical sequence of F that is valid and frequent. To find the valid subsequences of a candidate sequence s, we use the algorithm FindMaxSubseqLog with S containing only the initial state of $\mathcal{A}$ and E equal to the set of final states of $\mathcal{A}$.

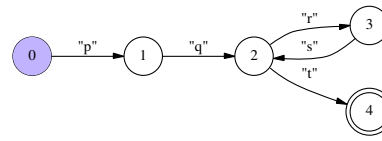


Fig. 2. DFA corresponding to the regular expression $pq(rs)^*t$

Stop Condition. For a step j of Algorithm 1, we stop if all the sets $F_j, \ldots, F_{j+|\mathcal{A}|-1}$ are empty ($|\mathcal{A}|$ is the number of states of $\mathcal{A}$). Indeed, if we consider a frequent valid logical sequence s of length greater than $j+|\mathcal{A}|-1$, then it must contains a loop of length at most $|\mathcal{A}|$ and thus a frequent valid subsequence of length equal at least to j. So, if no valid sequence of length greater or equal to j is frequent (since we supposed that $F_j, \ldots, F_{j+|\mathcal{A}|-1}$ are empty), then s cannot be frequent.

5 Experimental Results

We ran some experiments on the various versions of SPIRIT-LºG with data provided by a stochastic generator. This one begins by generating some types of constants and associates to each of them randomly generated constants. Then it generates predicate symbols of various arity, what allows to build ground atoms. Given a number of sequences to build and a maximum number of atoms per sequence, the generator can then build the database of input sequences.

For each variant, experiments show the impact of the different constraints and measure the trade-off between the support-based pruning and the pruning due to the use of the regular expression. As expected, results follow the same trend than those obtained with sequences of items [7]. However, we can remark that the use of logical sequences emphasizes some results. Indeed, except with very small automata, SPIRIT-LºG(R) becomes uninteresting, because the number of generated candidates quickly explodes. This is due to the fact that its candidate generation step often calls twice the Enum-Subst algorithm, which is generally costly in this variant. Like with SPIRIT, we can say that the weaker the relaxed constraint is, the more important the support-based pruning is and the less efficient becomes the pruning based on the use of the regular expression. Generally, SPIRIT-LºG(V) and SPIRIT-LºG(L) give the best trade-off between the both kinds of pruning. In fact, in many experiments, SPIRIT-LºG(L) was a little better than SPIRIT-LºG(V). It is because the number of renaming of variables used during the candidate generation step of SPIRIT-LºG(L) is generally smaller than the one used during the candidate generation step of SPIRIT-LºG(V) (In SPIRIT-LºG(L), we only have to consider renaming with variables of the first atom of the generated sequence, whereas in SPIRIT-LºG(V), we have to consider renaming with all variables occuring in the sequence to be extended).

On a database of 3000 sequences of maximum 50 elements each, and where arity of predicates was smaller than 3, we get curves of Figure 3. As expected, for low values of support, the number of candidates generated by SPIRIT-LºG(R) explodes. With a support of 0.1%, more than 150 000 candidate 6-sequences

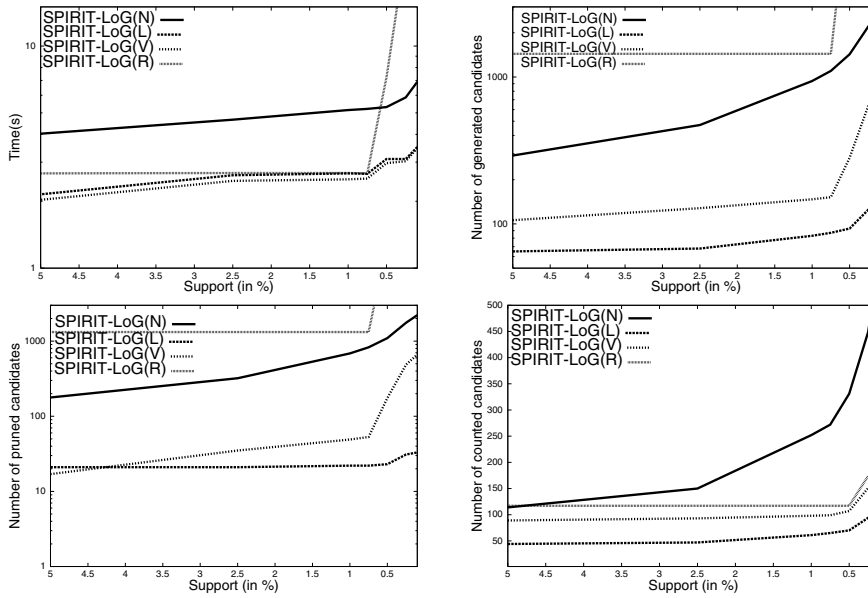


Fig. 3. Experimental results

were generated and we ran out of memory. SPIRIT-LºG(L) and SPIRIT-LºG(V) provide the best results. Despite a greater number of generate candidates, the pruning of SPIRIT-LºG(V) is very efficient. So, the number of candidates whose support is actually counted remains relatively similar for L and V variants.

Figure 3 shows that constraints are useful to increase the efficiency of SPIRIT-LºG. Indeed the number of candidates whose support is counted for SPIRIT-LºG(L) and SPIRIT-LºG(V) remains smaller than the one of SPIRIT-LºG(N).

6 Conclusion

In this work, we have presented a method for mining logical sequences from databases, and more particularly, we have choosen to adapt algorithms of [6] designed for the efficient mining of sequences of items.

The way we have proceeded to realize this adaptation of SPIRIT algorithms was mainly to adapt the generation and pruning functions of the different variants proposed by Minos N. Garofalakis. First, for the candidate generation step, the use of logical sequences involving variables leads us to consider the problem of variables renaming. It has an impact on the global behavior of the generation algorithm, which cost is directly linked to the number of candidates returned by the different calls to the Enum-Subst procedure. For instance, whereas with SPIRIT(L), the merge of two candidates leads only to one new candidate, the call to the Enum-Subst procedure in SPIRIT-LºG(L) can lead to many new candidates. Moreover, for the candidate pruning step, the adapation of the SPIRIT

algorithms mainly relies on the fact that the inclusion test between sequences of items has been replaced by an inclusion test between logical sequences that involves an unification test between variables of sequences.

Experimental results globally followed the same trend than results obtained with sequences of items, except for SPIRIT-LºG(R) whose candidate generation step is too costly. Another interesting point is that generally SPIRIT-LºG(L) is a bit better than SPIRIT-LºG(V) because, it calls the ENUM-SUBST function with sets of variables generally smaller than the ones used by SPIRIT-LºG(V).

The use of logical sequences containing variables also allows to introduce a new way of pruning redundancies in the final set of returned results. Indeed, for a given set of logical sequences that are all contained in the same examples of the input database, we can only return the most specific ones. For instance, if we have two logical sequences $s = \langle p_2(Z,T)\ p_4(Y,Z)\rangle$ and $t = \langle p_2(X,T)\ p_4(Y,Z)\rangle$ that are contained in the same examples of the input database, we can only return s because t is more general than s and so s contains more relational information relevant to the user.

The constraint we have choosen to consider here is also relevant with respect to related work in the field of logical sequence mining. Indeed, we introduce a syntactic constraint on mined logical sequence. However, this language bias does not have any impact on variable bindings when generating sequences. If we look at the language bias of WARMR, we remark that the `WRMode` allows to specify if a variable is strictly input, output, or both, what helps to reduce the number of generated candidates. Even if this bias is not as powerful as the use of regular expression, it allows to significantly reduce the search space by imposing a strong constraint on the possible variable bindings. So, our method is more useful for an user having an idea of predicate symbols occuring in the searched sequences, but having no particular knowledge of the possible bindings of variables of these predicates. However, WARMR is more intuitive for someone having a priori knowledge of the possible values of the parameters of predicate symbols (so, he can express that he wants to mine sequences where the first argument of the 'lpr' predicate is the last argument of the 'dvips' predicate). Integrating such a constraint in SPIRIT-LºG could be of great help to reduce the computational cost of generating functions, because the number of generated candidates with the different variants of SPIRIT-LºG is linked to the arity of the atoms and to the fact that all variables of an atom can be renamed in any other variable. Thus, considering both a syntactic constraint and a constraint on variables bindings could more reduce the number of generated candidates.

Finally, there are many perspectives to our work. First, we wish to study further constraints so as to more reduce the search space. For instance, integrating both syntactical constraint provided by SPIRIT-LºG and a constraint on variables bindings like the `WRMode` would be very useful. Moreover, we wish to extend SPIRIT-LºG in order to take into account a background knowledge while extracting frequent logical sequences. We could then extract couples (s, c) of logical sequences s linked to some conditions c expressed under the form of Horn clauses. Thus, it would be possible to say that we only want to extract

logical sequences in input sequence for which a Horn clause is true. For instance, let us suppose that an input sequence containing 'emacs' and 'gcc' commands is characteristic of expert users of UNIX systems, we can then extract only logical sequences in input sequences corresponding to expert users. Another extensions that could be done to our algorithms is to extract logical sequences made up of atoms involving both variables and constants and not only variables. In our first implementation, SPIRIT-LºG can only extract logical sequences containing variables, but it is clear that extracting atoms containing some constants could help to improve the efficiency of our prototype. Indeed, at the candidate generation step, it would not be useful to consider the renaming of a parameter that has already been transformed in a constant and so the search space will be reduced. Finally, the counting step being the more expensive one, we want to work on this problem in order to incorporate some optimizations on that point into SPIRIT-LºG.

Acknowledgements

The authors wish to thank Jean-François Boulicaut for his help during the writing of this paper and the financial support for the presentation of this work at the ILP'02 conference.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Intl. Conf. on Very Large Databases*, pages 487–499, Santiago de Chile, Chile, September 1994.
2. R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of the 11th Intl. Conf. Data Engineering*, pages 3–14, Taipei, Taiwan, March 1995.
3. H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, June 1998.
4. L. Dehaspe and H. Toivonen. Discovery of frequent Datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.
5. S. Dzeroski and N. Lavrac, editors. *Relational Data Mining*. Springer-Verlag, 2001.
6. M. N. Garofalakis, R. Rastogi, and K. Shim. Spirit : Sequential pattern mining with regular expression constraints. Technical report, Bell Laboratories, 1999.
7. M. N. Garofalakis, R. Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression constraints. In *VLDB'99, Proc. of 25th Intl. Conf. on Very Large Databases*, pages 223–234, Edinburgh, Scotland, UK, September 1999.
8. N. Jacobs and H. Blockeel. From shell logs to shell scripts. In *Proc. of the 11th Intl. Conf. on Inductive Logic Programming*, pages 80–90, Strasbourg, France, September 2001.
9. S. D. Lee and L. De Raedt. Constraint based mining of first-order sequences in seqlog. In *Proc. of the WorkShop on Multi-Relational Data Mining, colocated with ACM SIGKDD'02*, Edmonton, Alberta, Canada, July 2002.
10. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *Proc. of the 1st Intl. Conf. on Knowledge Discovery and Data Mining*, pages 210–215, Montreal, Canada, August 1995.

11. C. Masson and F. Jacquenet. D couverte de s quences logiques fr quentes sous contraintes. In *Actes du 13ème Congrès Francophone de Reconnaissance des Formes et Intelligence Artificielle*, pages 673–684, Angers, France, January 2002. In French.
12. R.T. Ng, V.S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, pages 13–24, Seattle, Washington, USA, June 1998.
13. R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. of the 5th Intl. Conf. Extending Database Technology*, pages 3–17, Avignon, France, March 1996.
14. M.J. Zaki. Sequence mining in categorical domains: Incorporating constraints. In *Proc. of the 9th Conf. on Information and Knowledge Management*, pages 422–429, McLean, VA, November 2000.

Using Theory Completion to Learn a Robot Navigation Control Program

Steve Moyle

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, England

Abstract. An Event Calculus program to control the navigation of a real robot was generated using Theory Completion techniques. This is an application of ILP in the non-observational predicate learning setting. This work utilized 1) extraction-case abduction; 2) the simultaneous completion of two, mutually related predicates; and 3) positive observations only learning. Given time-trace observations of a robot successfully navigating a model office and other background information, Theory Completion was used to induce navigation control programs in the event calculus. Such programs consisted of many clauses (up to 15) in two mutually related predicates. This application demonstrates that abduction and induction can be combined to effect non-observational multi-predicate learning.

1 Introduction

The Event Calculus is a logic programming framework for reasoning about actions and change. Its simple axioms encode the common sense law of inertia. It consists of the following components: the core axioms (which provide the law of inertia), the domain specific axioms (which are provided by a programmer), an initial state of the system, and a sequence of events. From these it is possible to reason about the state of the system at different times.

Previous work on learning Event Calculus programs [Moyle, 1999] suffered from the following constraints: 1) Only a single predicate symbol could be induced at one time (this was overcome by combining predicates into a meta predicate); 2) negative observations were required to be artificially produced; and 3) the application domain was very restricted.

The Event Calculus has been used to program autonomous robots capable of navigating in a simple model office. One of the components of the robot's event calculus program is used for controlling the navigation of the robot. This component converts a sequence of events or actions into outputs that guide the robot. Other researchers (e.g. [Klingspor and Morik, 1995]) have used first-order representations for robot programming (and learning), however this work focuses on techniques utilizing the Event Calculus.

The Event Calculus, as a reasoning framework, can be utilised in three alternative ways. Previous researchers have shown how an event calculus program can be used *deductively* to predict how the state of the program evolves over time

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 182–197, 2003.

(e.g. [Shanahan, 1989]). Also work has shown how an event calculus program can be used *abductively* to plan the sequence of events necessary to change the system from an initial state into a goal state (e.g. [Kakas and Michael, 1995]).

The work reported in this paper focuses on a third, *inductive* way of using an event calculus program. Here the objective is to induce the domain specific axioms – the situation-action rules – from observations of the evolving states of a system.

The technique used for such induction is Theory Completion, which is an extension to traditional Inductive Logic Programming. The form of Theory Completion used in this paper combines both abduction (utilizing Yamamoto's SOLD resolution [Yamamoto, 2000]) and induction.

The experiments performed on the robot domain illustrate that it is possible to re-generate a robot navigation control program (in the event calculus) that contains a significant number of clauses (15).

The remainder of the paper is organised as follows. Section 2 introduces the event calculus and the robot navigation control program. Section 3 outlines Theory Completion in Inductive Logic Programming. In section 4, an experiment to re-generate the robot navigation control program is described, and the results are presented in section 5. The results are discussed in the final section of the paper.

2 An Event Calculus Program to Control the Navigation of a Robot

2.1 The Event Calculus

The Event Calculus was developed as a general approach to representing and reasoning about events and their effects in a logic programming framework, and as an attempt to solve the frame problem [Kowalski and Sergot, 1986]. An Event Calculus program allows a dynamic system to be modelled so that future states of the system can be determined, thus enabling the derivation of system values that *hold at* future times based upon events that have *happen*ed at previous times.

The Event Calculus utilises the *common sense law of inertia* [Shanahan, 1997] also known as *default persistence*. This law formalises the notion that properties remain unchanged throughout time unless effected by the action of an event. Change is mediated solely through the action of events, where events are related to system properties if they initiate (or terminate) periods in which the properties hold (or do not hold). Default persistence allows reasoning about properties even when information about the world is not precisely known.

A burden of all event calculi is that they require humans to provide domain specific axioms (DSAs) for definitions of predicates that encode the dynamics of the domain - a process that can be difficult. This difficulty is exacerbated by the non-monotonicity of the formalism[1]. One way to reduce this burden would

[1] The non-monotonicity is caused, in part, by the use of Negation-as-failure.

be to induce predicate definitions for the domain specific axioms from observed time-traces of property values from an existing dynamic system.

The Event Calculus has been used to describe several domains including an Air Traffic Control system [Sripada, 1995]. Researchers in Leuven have successfully implemented a sliding window protocol with go-back-n in the Event Calculus [Denecker *et al.*, 1996], from which they were able to prove that the communications system as modelled was deadlock free. The Event Calculus has also been used as the basis for solving planning problems (e.g. [Kakas and Michael, 1995]).

2.2 An Event Calculus Program to Control a Robot's Navigation

Shanahan and colleagues [Shanahan, 2000] have been studying robots which are small (55mm diameter, and 30mm high), and are relatively low cost. Their motion is provided by three wheels, which are driven by two DC motors. They have sensory input from eight infra-red proximity and light sensors. The robots are programmed in a novel high-level language: a version of the *E*vent Calculus. The robot, suitably programmed, is capable of moving within the confines of a "model office". The layout of the office is shown in figure 1. The actual model office is constructed from wood with an overall floor plan of 3 m by 6 m (the walls are 10 mm high).

The robot's event calculus program contains a portion devoted to navigational control. This portion consists of situation-action rules that specify when the robot's actions will be successful. For instance that the robot may turn left into a doorway, provided the robot is at the doorway and the door is open.

There are many aspects of robot control that the researchers have studied: these include *planning* and *navigating*. Planning typically produces an ordered set of *actions* to move the robot from its initial position to some desirable final position. Navigating can be viewed as the execution of the *planned* set of *actions*. The program that takes the actions and the robot's start position, and then controls the robot to its destination is known here as the **Robot Navigation Control program**.

The Components of the Robot Navigation Control Program. The Robot Navigation Control Program is made up of the "usual" elements needed for an Event Calculus program: core axioms, domain specific axioms, an initial state, a narrative of events, and other background information. In what follows, the version of the Event Calculus that is used is known as the *Simplified Event Calculus* [Sadri and Kowalski, 1995].

Core Axioms. The core axioms for the Robot Navigation Control program are presented in figure 2. The meaning of these axioms is that i) a property (or fluent) remains unchanged from its initial value if it is not clipped in the time period; and ii) a fluent value can be ascertained from the the previous occurrence of any event that initiates a change in the value, provided that this value has not been clipped since. Finally, a fluent value is clipped in some time period if an event happens in the period that terminates the fluent value.

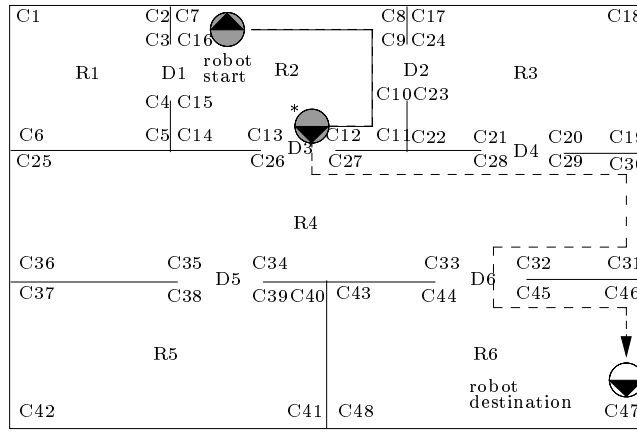


Fig. 1. A schematic of the robot and its "model office" environment. The robot is represented by a circle with a triangle inside. The right angle of the triangle represents the direction the robot is facing. Shaded robots indicate that the robot is in transit. The unshaded robot is stationary. The robot's initial position and destination are shown. The robot at $*$ is enroute to the final destination. The office consists of *rooms* - *R*, *doors* - *D*, and *corners* - *C*.

Domain Specific Axioms. These axioms or "rules" are situation-action rules that specify what actions – or events – cause what changes in state, provided that the preconditions are met when the event occurs. There are two fundamental changes in state that can occur: initiation and termination of states. Typically the domain specific axioms referring to a particular event trigger have rules for both initiation and termination. A portion of the domain specific axioms that control the robot's navigation are listed in figure 8.

The robot's navigation strategy can be informally described as "cling-to-the-wall" and as such the robot is restricted to performing a set of primitive actions. These actions (or events) are described in table 1. Notice that these rules can be applied to different situations – for instance alternative "room" layouts, different event narratives, and different initial situations. The first domain specific axiom shown in figure 8 defines the *initiating* relationship between the *follow_wall* action and the robot's *loc*ation fluents.

$$initiates(follow\_wall, loc(corner(C_2), ahead), T) \leftarrow$$
$$holds\_at(loc(corner(C_1), behind), T) \wedge$$
$$next\_visible\_corner(C_1, C_2, left, T).$$

This axiom states that "*the occurrence of a follow_wall event causes the robot to move to a location where the next visible corner on the left is immediately ahead – provided that the robot is located at a corner immediately behind – at the time the event occurs*".

The second domain specific axiom shown in figure 8 defines the *terminating* relationship between the *follow_wall* action and the *loc*ation fluent.

```
holds_at(F,T):-   initially(F), not clipped(0,F,T).
holds_at(F,T2):-   happens(A,T1), lt(T1,T2),
     initiates(A,F,T1), not clipped(T1,F,T2).

clipped(T1,F,T2):- happens(A,T), le(T1,T),
     lt(T,T2), terminates(A,F,T).
```

Fig. 2. The Event Calculus core axioms. Predicates `lt/2` and `le/2` represent less-than and less-than-or-equal to.

Initial Situation. An example initial state for the robot is presented in figure 7 (refer also to figure 1). It is a set of special fluent values, stating the location of the robot, and open-closed state of the doors in the office at time point 0.

Narrative of Events. The narrative of events is the temporally ordered set of actions that, when correctly "applied", will enable the robot to navigate from its initial state to its destination. An example narrative[2] is shown in figure 7. In this example, the fact $happens(turn(left), 27)$ states that at time point 27 the action $turn(left)$ occurs.

Other Background Information. The floor plan of the "building" that the robot navigates within is presented in figure 1. The description of such a plan is provided to the Event Calculus in the form of a PROLOG program. A component of such a plan relating to room 2 is shown in figure 7.

The complete robot control program (and its physical embodiment) accounts for the fact that the doors in the environment can open or close at anytime. The robot, however, can only deduce that a door is closed by recognising that by following the wall it gets to the next corner, and *not* a doorway. This deduction is provided via the clauses in figure 7.

The robot navigation control program contains 26 non-unit clauses, 275 unit clauses, with a total number of 358 literals. This does not include the domain specific axioms which contain 15 clauses and 73 literals.

Executing the Robot Navigation Control Program. Having assembled all the components of the Robot Navigation Control (RNC) program, what can it be used for? Being a PROLOG program, the RNC may be interpreted by a PROLOG engine. With the RNC loaded it is possible to query the system as to the value of any fluents at any particular time.

For example, to find out what the state of the world is at time '*' in figure 1, the PROLOG engine can be queried[3] for `holds_at(Fluent,26)`, which returns the following fluent values: `in(r2)`, `loc(door(d3),in)`, `door_open(d1)`, `door_open(d2)`, `door_open(d3)`, `door_open(d4)`, `door_open(d5)`, and `door_open(d6)`. The meaning of these bindings for the fluent is that at time

[2] This narrative was generated by the Abductive Event Calculus Planner as described in [Shanahan, 2000].

[3] This is an illustration of utilizing an event calculus program for *prediction*.

Table 1. The primitive actions of the Robot.

Action	Description	Pre-condition	Post-condition
turn(right)	Change the robot's direction $+\pi/2$	corner is ahead	corner is behind
turn(left)	Change the robot's direction $-\pi/2$	corner is ahead	corner is behind
follow_wall	Move the robot along the wall to the next corner	corner is behind	corner is ahead
go_straight	Move across an (open) doorway	doorway ahead	doorway behind
turn(left)	Move into a doorway	standing at a corner to an open door	standing in the door
turn(left)	Move into the adjacent room	standing in a connecting doorway	standing in the next room

point 26 all the doors are open, and the robot is located in door *D3*, having approached the door from room *R2*.

Further examples of the fluent values that can be derived by assembling all the components of the Robot Navigation Control program are shown in figure 6.

3 Theory Completion in Inductive Logic Programming

Traditional explanatory ILP [Muggleton, 1995] generalises instances (often ground) of example phenomena. Typically the examples are described by the same predicate as that of their generalization (or clauses). This is known as *observational predicate learning* or OPL [Muggleton and Bryant, 2000].

OPL is inadequate for learning Event Calculus domain specific axioms from fluent time-trace observations. Consider that the fluent observations are recorded in the predicate symbol `holds_at/2` while the clauses that describe the situation-action rules are in two different predicate symbols: `initiates/3` and `terminates/3`. To overcome this deficiency a form of non-OPL ILP is required called Theory Completion [Moyle and Muggleton, 1997].

In Theory Completion a combination of *abduction* and *induction* can be applied to perform non-OPL. First, the examples (in one predicate) are transformed, with respect to the background program, into a set of abduced explanations in terms of other predicates. These abduced explanations can then be generalised by normal ILP techniques.

3.1 Generating Abductive Explanation

The process of transforming examples in one predicate to explanations in terms of other predicates is known as *extraction case abduction* [Kakas and Riguzzi, 2000]. Many abductive techniques have been proposed, but for this work Yamamoto's SOLD resolution [Yamamoto, 2000] is used.

Consider the fluent `holds_at(in(r4),28)` from the fluent trace from the Robot Navigation Control program as listed in figure 6. This fluent value has changed since the previous time point where its value was `holds_at(loc(door(d3),in),27)`. Abduction can be used to extract that one explanation for this change is possibly due to the event occurrence `happens(turn(left),27)` giving rise to the changes `initiates(turn(left),in(r4),27)` and `terminates(turn(left), loc(door(d3),in),27)`.

3.2 Theory Completion with ALECTO

Abduction and induction are combined in the ILP system ALECTO [Moyle, 2000]. ALECTO is an extension of *Inverse Entailment and Progol* [Muggleton, 1995] and is implemented as an add-on to the ALEPH [Srinivasan, 2000]. It uses extraction-case abduction, and only requires *positive* examples.

ALECTO's implementation is based on the following setting.

- *Given*
  Background knowledge (or incomplete theory) as a set of Horn clauses: $B = \{C_1, C_2, \ldots\}$
  Examples as a set of (ground) facts: $E^+ = \{e_1^+, e_2^+, \ldots\}$
- *Do ...*
  For *each* example e_i^+ generate an abductive explanation (ground) Δ_i from e_i^+ and B
  Collect the explanations for *all* examples into the **start set** $\chi = \bigcup \Delta_i$ (Note that $B \cup \chi \models E^+$)
  Generalise χ to produce completed theory H.
- *Output*
  A set of clauses to complete the theory H, such that $B \cup H \models E^+$.

In this process ALECTO uses abduction in the following manner. From one positive example e^+ and the incomplete background theory B, sets of **seed** atoms are *abduced* using Yamamoto's *SOLD resolution*[4]. Consider the case when B is the Robot Navigation Control event calculus program without DSAs and $e^+ = holds\_at(in(r4), 28)$ then

$SOLDR(B, e^+) = F$
$= \{initiates(turn(left), in(r4), 27), terminates(turn(left), loc(door(d3), in), 27)\}$.

Note that, in general, SOLDR may produce *non-unique* and *non-ground* atom sets. In some circumstances this can be overcome by constraining the abduction process with "declarative bias".

ALECTO's generalisation process is similar to the cover-set algorithms of other ILP systems (e.g. [Srinivasan, 2000]). For every atom a_j (ground) from the start-set χ

[4] SOLDR = **S**kip **O**rdered **L**inear resolution for **D**efinite clauses. A resolution technique for deriving a goal-clause $\overline{S}$ from a definite program B and a goal $\overline{G}$ where $B \cup \overline{G} \models \overline{S}$.

– Produce a "bottom" clause $\perp_{a_j}$ by mode directed inverse entailment. e.g. $\perp_{terminates}=$

```
terminates(turn(left),loc(door(A),in),B):-
   holds_at(facing(C),B), holds_at(in(D),B), door(A,G,H),
   holds_at(loc(door(A),in),B), holds_at(door_open(A),B), door(E,M,N),
   holds_at(door_open(E),B), door(A,I,J), holds_at(door_open(F),B),
   door(F,O,P), door(E,K,L), door(F,Q,R).
```

– Search the θ-subsumption lattice for the "best" clause H_{a_j} where $\square \succeq H_{a_j} \succeq \perp_{a_j}$.

There is one main departure from the ALEPH algorithm when clauses are scored due to the nature of abduction during the search for multiple predicates. ALECTO performs a multi-predicate search – one predicate at time. This requires the following assumption: *the abduced atoms, yet to be entailed by the theory so far, are "true" while searching for new clauses to add to the theory.* From here ALECTO uses a *Pos-only* [Muggleton, 1996] style function for scoring to trade-off the *generality* of the clauses induced against *size* of the completed theory. The *generality* of the clauses is measured against the *example space* (for the event calculus this is the observations of holds_at/2 predicates), while theory size is an estimate of the *final theory size* (measured in literals), and includes both clauses and the un-covered start-set atoms.

4 An Experiment to Learn a Robot Navigation Control Program

The objective of the experiments was to test the hypothesis that: *Theory Completion techniques can be used to induce clauses for domain specific axioms of Event Calculus programs from observations of a dynamic system.* In particular, the aim was to induce domain specific axioms (DSAs) for the navigation control of the robot as described in section 2.2.

The general procedure that was used for the controlled experiment can be broadly categorised into two tasks: 1) data generation; and 2) induction. The data generation was performed by using an existing, correct event calculus program and executing it to generate observations of the program's dynamic behaviour. This execution is a deductive process which produces time-trace data. The time-trace data is then used as input to the Theory Completion system ALECTO.

The series of experiments for generating domain specific axioms for the robot navigation control program were performed in the following manner. The experimental method consisted of: 1) randomly generating trace data; 2) generating DSAs using the data with ALECTO; and 3) testing the performance of the generated DSAs on multiple test sets. For these experiments the data for thirty "runs" were generated from thirty pairs of randomly selected start and end corners in the model office.

For each run, a plan – a sequence of events – was constructed that would enable the robot to navigate from the start corner to the end corner (or goal

```
terminates(turn(right),loc(corner(A),ahead),B):-
  holds_at(loc(corner(A),ahead),B), inner(A).
terminates(turn(A),facing(B),C) :-
  holds_at(facing(B),C),change_direction(A,D,B),
  direction(D), direction(B).
terminates(turn(left),loc(door(A),in),B) :-
  holds_at(loc(door(A),in),B),
  holds_at(door_open(C),B), door(C,D,E),
  corner(D), corner(E).
terminates(turn(left),loc(corner(A),ahead),B):-
  holds_at(loc(corner(A),ahead),B), corner(A).
terminates(turn(left),in(A),B) :-
  holds_at(in(A),B),holds_at(loc(door(C),in),B),
  door(C,D,E), corner(D), corner(E).
terminates(go_straight,loc(corner(A),ahead),B):-
  holds_at(door_open(C),B), door(C,A,D),
  corner(D),holds_at(loc(corner(A),ahead),B).
terminates(follow_wall,loc(corner(A),behind),B):-
  holds_at(loc(corner(A),behind),B), corner(A).

initiates(go_straight,loc(corner(A),behind),B):-
  holds_at(door_open(C),B), door(C,D,A),
  holds_at(loc(corner(D),ahead),B).
initiates(turn(right),loc(corner(A),behind),B):-
  holds_at(loc(corner(A),ahead),B), inner(A).
initiates(turn(A),facing(B),C) :-
  holds_at(facing(D),C),
  holds_at(door_open(E),C),
  door(E,F,G), corner(F), corner(G),
  change_direction(A,D,B).
initiates(turn(left),loc(door(A),in),B) :-
  holds_at(door_open(A),B),
  door(A,C,D), corner(D),
  holds_at(loc(corner(C),ahead),B).
initiates(turn(left),loc(corner(A),behind),B):-
  holds_at(facing(C),B), direction(C),
  holds_at(loc(door(D),in),B),
  door(D,E,A), corner(E).
initiates(turn(left),in(A),B) :-
  holds_at(in(C),B),
  holds_at(loc(door(D),in),B), door(D,E,F),
  corner(E), corner(F), connects(D,C,A).
initiates(follow_wall,loc(corner(A),ahead),B):-
  holds_at(door_open(C),B), door(C,A,D),
  next_visible_corner(D,E,B), corner(E),
  holds_at(loc(corner(F),behind),B), corner(F).
initiates(follow_wall,loc(corner(A),ahead),B):-
  holds_at(loc(corner(C),behind),B),
  next_visible_corner(C,A,B).
```

Fig. 3. Sample DSAs generated by ALECTO for the robot navigation control program.

location). The complete Event Calculus program was then run deductively to produce fluent traces. All 15 of the the domain specific axioms were then removed and the remaining portions of the program was input to ALECTO, which produced a set of clauses.

Each of the thirty generated DSAs was evaluated on the other 29 traces. This evaluation was based on the number of correctly predicted transitions by the induced DSAs compared with those of the original DSAs. For each DSA set an accuracy on each of the other 29 traces was produced, and these were averaged to produce a mean accuracy and variance for the set.

4.1 Results

The Theory Completion techniques were able to induce sets of DSAs for the RNC, with an average number of 14 clauses in the 30 DSAs sets generated. The estimated average accuracy of the induced DSAs over the 30 runs was 70.2%, with an average of the standard deviations within each run being 13.1%.

One example of a set of clauses produced by ALECTO is the DSAs shown in figure 3. Murray Shanahan, the originator of the RNC, was then asked to review the automatically generated DSAs. In his review he comments

> All in all, since the system only gets a snapshot of cases to generalise from, I'm quite impressed.

The way the accuracy of the generated DSAs varies with the number of abducible observations (these are related to the number of events that act on the robot's trace) is presented in figure 4. The run time of the generation of

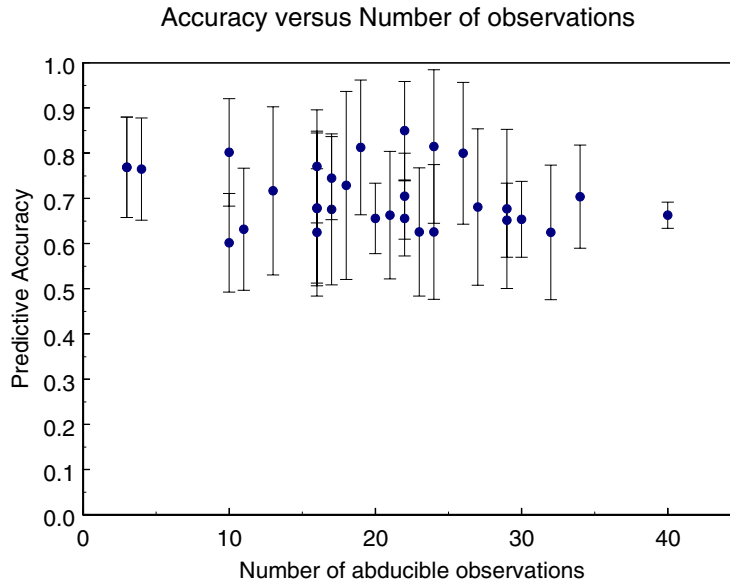


Fig. 4. Accuracy versus the number of abducible observations (related to the number of events that act on the robot's trace). Note that two data items appear overlaid at the abscissa value of 16 – 67.81 ± 17.14% and 67.90 ± 16.57%.

the DSAs is presented in figure 5. This includes both the contribution from the extraction-case abduction and the induction.

5 Discussion

The experiments demonstrate that Theory Completion by `ALECTO` can generate 'reasonable' domain specific axioms for the robot navigation control program. The size of the induced programs is quite high (the average number of clauses produced across the 30 runs was 14).

The clauses produced by Theory Completion are not in the same predicate symbol as those of the examples. The Event Calculus domain specific axioms produced are in terms of `initiates/3` and `terminates/3` while the examples are in terms of `holds_at/2`. This is an example of *non-observational predicate learning*. Furthermore, the clauses produced are in two mutually-related predicates in the Event Calculus. This is an example of completing more than one predicate at a time (a form of multiple-predicate learning (e.g. [De Raedt *et al.*, 1993] and [Esposito *et al.*, 2000]) – albeit in a coarsely interleaved manner).

The accuracy of the DSAs produced does not appear to vary markedly with the number of examples (see figure 4). Here an example can be considered to be an observation that has an abductive explanation with respect to both the background theory (Event Calculus core axioms) and the declarative bias – i.e. a `happens/2` event that can explain the change in a fluent value. The most

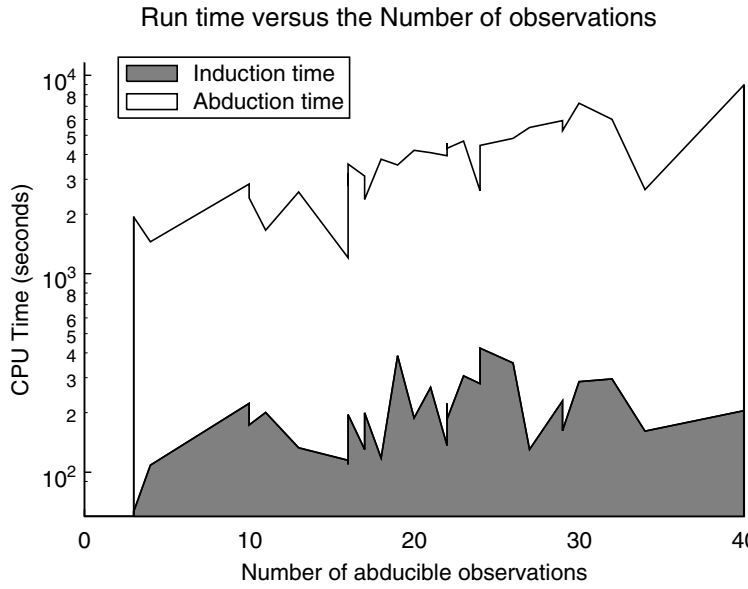


Fig. 5. The run time of both the extraction-case abduction and the induction versus the number of abducible observations. The experiments were performed on hardware using Intel Pentium III 550MHz processors, and 512 MB RAM.

significant component of learning run times for the domain specific axioms is that associated with the extraction-case abduction (see figure 5). This is due to the manner in which the cross-lists were generated to minimise the amount of intersection of abducibles within them. The formulation of the Event Calculus exacerbated this as any member of a sequence of stable fluent values (i.e. observations) can be utilised by SOLD to produce the same abducible atom.

6 Conclusions

This paper has presented a framework for the automatic generation of non-trivial Event Calculus programs from observations of dynamic system behaviour. This framework utilises Theory Completion techniques in Inductive Logic Programming. This extends earlier and overcomes previous deficiencies by incorporating the completion of two related predicates; learning from positive-only observations; and the application to a non-trivial Event Calculus domain.

Learning the domain specific axioms for an Event Calculus program completes the use of three modes of logical reasoning suggested by philosopher Peirce [Peirce, 1931]. An Event Calculus program can be used *deductively* (core axioms plus domain specific axioms plus narrative of events) to predict *where* the robot will be at future times. It can be used *abductively*[5] (core axioms plus domain

[5] For a comprehensive discussion on *abduction* see [Flach and Kakas, 2000].

specific axioms plus start state and desired end state) to provide a *plan* (a narrative of events). Finally, it can be used *inductively* (core axioms plus time-trace of where the robot is plus a narrative of events) to produce rules that explain *why* the robot behaves it the way it does.

Future work should consider the following topics: new application domains, noise, and extensions to the reasoning system. Applying the framework to different application domains in the Event Calculus (e.g. learning DSAs for modelling ship damage control [Bulitko and Wilkins, 1999]). The abductive explanations in this work have been generated using accurately classified observations. In what ways would the introduction of *noise* alter the setting? What noise handling techniques would be best suited to this problem?

Future work could also explore how problems which can be described using continuous change extensions to the event calculus [Shanahan, 1990] could be solved. Also those domains where time lags exist between the occurrence of the initiating/terminating event and the change in fluent value. Finally, it may be possible to learn rules about action and change using an analogous logic programming implementation of the situation calculus [McCarthy, 1959] which has also been the focus of Lorenzo and Otero [Lorenzo and Otero, 2000].

Acknowledgements

The author gratefully acknowledges Murray Shanahan for the use of his robot system. This worked benefited from the generous assistance and encouragement of Ashwin Srinivasan.

The work reported here was supported in part by the EU project Sol-EU-Net, IST-11495.

References

Bulitko and Wilkins, 1999. V.V. Bulitko and D.C. Wilkins. Learning to Envision: An Intelligent Agent for Ship Damage Control. In *Proceedings of the ACAI'99 conference*, Crete, Greece, 1999.

De Raedt *et al.*, 1993. Luc De Raedt, Nada Lavrac, and Saso Dzeroski. Multiple predicate learning. In *Preceedings of IJCAI 93*, p. 1037 - 1042. 1993.

Denecker *et al.*, 1996. Marc Denecker, Kristof Van Belleghem, Guy Duchatelet, Frank Piessens, and Danny De Schreye. A Realistic Experiment in Knowledge Representation in Open Event Calculus: Protocol Specification. In M. Maher, editor, *Proceedings of JICSLP'96, the Joint International Conference and Symposium on Logic Programming*, p. 170-184. Bonn, Germany, 1996. MIT Press.

Esposito *et al.*, 2000. Floriana Esposito, Donato Malerba, and Francesca A. Lisi. Induction of Recursive Theories in the Normal ILP Setting: Issues and Solutions. In J. Cussens and A. Frisch, editors, *Inductive Logic Programming: 10th International Conference*, p. 93 - 111. Springer, London, 2000.

Flach and Kakas, 2000. Peter A. Flach and Antonis C. Kakas eds. *Abduction and Induction:Essays on their Relation and Integration.* Applied Logic Series, D. M. Gabbay. Vol. 18. Kluwer Academic Publishers, 2000.

Kakas and Michael, 1995. A. C. Kakas and A. Michael. Integrating Abductive and Constraint Logic Programming. In *Proceedings of the 12th International Conference on Logic Programming*, Japan, 1995.

Kakas and Riguzzi, 2000. Antonis C. Kakas and Fabrizio Riguzzi. Abductive Concept Learning. *New Generation Computing*, 18:243-294, 2000.

Klingspor and Morik, 1995. Volker Klingspor and Katharina Morik. Towards Concept Formation Grounded on Perception and Action of a Mobile Robot. In U. Rembold, et al., editors, *IAS-4, Proceedings of the 4th International Conference on Intelligent Autonomous Systems*, IOS Press, 1995.

Kowalski and Sergot, 1986. Robert Kowalski and Marek Sergot. A Logic-based Calculus of Events. *New Generation Computing*, 4:67 - 95, 1986.

Lorenzo and Otero, 2000. David Lorenzo and Ramon P. Otero. Using a ILP algorithm to learn Logic Programs for Reasoning about Actions. In J. Cussens and A. Frisch, editors, *Inductive Logic Programming: Work-in-Progress Reports*, p. 163 - 171. London, 2000.

McCarthy, 1959. John McCarthy. Programs with Common Sense. In *Proceedings of the Symposium on Mechanisation of Thought Processes*, London, 1959. Her Majesty's Stationery Office.

Moyle and Muggleton, 1997. S. Moyle and S. Muggleton. Learning Programs in the Event Calculus. In N. Lavrac and S. Dzeroski, editors, *7th International Workshop, ILP-97*, p. 205–212. Prague, 1997. Springer.

Moyle, 1999. S. A. Moyle. Learning about Action and Change: An Inductive Logic Programming approach. Technical report, Oxford University Computing Laboratory, University of Oxford, Oxford, 1999.

Moyle, 2000. S. A. Moyle. *An investigation into Theory Completion techniques in Inductive Logic Programming*. Thesis for D Phil, Oxford University Computing Laboratory, University of Oxford, 2000.

Muggleton, 1995. S. Muggleton. Inverse Entailment and Progol. *New Generation Computing*, 13(3 and 4):245–286, 1995.

Muggleton, 1996. S. Muggleton. Learning from positive data. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, p. 225-244. 1996.

Muggleton and Bryant, 2000. Stephen Muggleton and Christopher Bryant. Theory Completion Using Inverse Entailment. In J. Cussens and A. Frisch, editors, *Inductive Logic Programming: 10th International Conference*, p. 130 - 146. London, 2000. Springer.

Peirce, 1931. Charles Sanders Peirce. Elements of logic. In C. Hartshorne and P. Weiss, editors, *Collected papers of Charles Sanders Peirce*, Harvard University Press, Cambridge, Massachusetts, 1931.

Sadri and Kowalski, 1995. Fariba Sadri and Robert Kowalski. Variants of the Event Calculus. In *Proceedings of the 12th International Conference on Logic Programming*, Japan, 1995.

Shanahan, 1989. Murray Shanahan. Prediction is Deduction but Explanation is Abduction. In *Preceedings of IJCAI 89*, p. 1055 - 1060. 1989.

Shanahan, 1990. Murray Shanahan. Representing Continuous Change in the Event Calculus. In *Proceedings of ECAI 90*, p. 598-603. 1990.

Shanahan, 1997. Murray Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, 1997.

Shanahan, 2000. Murray Shanahan. A Logical Account of the Common Sense Informatic Situation for a Mobile Robot. *Electronic Transactions on Artificial Intelligence*, , 2000.

Srinivasan, 2000. Ashwin Srinivasan, The Aleph Manual 2000, `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/`.

Sripada, 1995. Suryanarayana M. Sripada. Efficient Implementation of the Event Calculus for Temporal Database Applications. In *Proceedings of the 12th International Conference on Logic Programming*, Japan, 1995.

Yamamoto, 2000. Akihiro Yamamoto. Using abduction for induction based on bottom generalization. In P. A. Flach and A. C. Kakas, editors, *Abduction and Induction: Essays on their Relation and Integration*, Kluwer Academic Publishers, 2000.

Appendix

```
holds_at(loc(corner(c7),ahead),0).
holds_at(in(r2),0).
holds_at(loc(corner(c7),ahead),1).
holds_at(in(r2),1).
           :
holds_at(loc(corner(c12),ahead),25).
holds_at(in(r2),25).
holds_at(loc(door(d3),in),26).
holds_at(in(r2),26).
```

```
holds_at(loc(door(d3),in),27).
holds_at(in(r2),27).
holds_at(loc(corner(c27),behind),28).
holds_at(in(r4),28).
           :
holds_at(loc(corner(c46),behind),65).
holds_at(in(r6),65).
holds_at(loc(corner(c47),ahead),66).
holds_at(in(r6),66).
```

Fig. 6. Example fluent values at different time points as derived from the Robot Navigation Control program.

```
/* Initial Situation */
initially(door_open(d1)).       initially(door_open(d3)).
initially(door_open(d4)).       initially(door_open(d6)).
initially(door_open(d2)).       initially(loc(corner(c7),ahead)).
initially(door_open(d5)).       initially(in(r2)).

/* Narrative of Events */       happens(follow_wall,29).
happens(turn(right),4).         happens(go_straight,31).
happens(follow_wall,5).         happens(follow_wall,33).
happens(turn(right),9).         happens(turn(right),39).
happens(follow_wall,11).        happens(follow_wall,43).
happens(go_straight,13).        happens(turn(right),49).
happens(follow_wall,17).        happens(follow_wall,53).
happens(turn(right),19).        happens(turn(left), 55).
happens(follow_wall,23).        happens(turn(left), 57).
happens(turn(left), 25).        happens(follow_wall,59).
happens(turn(left), 27).        happens(turn(right),62).
                                happens(follow_wall,65).
/* Room 2 layout */
next_corner(r2,c7,c8).         door(d1,c15,c16).
next_corner(r2,c8,c9).         door(d2,c9,c10).
next_corner(r2,c9,c10).        door(d3,c12,c13).
next_corner(r2,c10,c11).       door(d3,c26,c27).
next_corner(r2,c11,c12).
next_corner(r2,c12,c13).       inner(c7).
next_corner(r2,c13,c14).       inner(c8).
next_corner(r2,c14,c15).       inner(c11).
next_corner(r2,c15,c16).       inner(c14).
next_corner(r2,c16,c7).
next_corner(r4,c26,c27).       connects(d1,r1,r2).
next_corner(r4,c27,c28).       connects(d2,r3,r2).
                               connects(d3,r2,r4).
/* Miscellaneous */
next_visible_corner(C1,C2,left,T):- holds_at(in(R),T),
    next_corner(R,C1,C2), not(invisible_corner(C2,T)).
next_visible_corner(C1,C3,left,T):- holds_at(in(R),T),
    next_corner(R,C1,C2), invisible_corner(C2,T),
    next_visible_corner(C2,C3,left,T).

invisible_corner(C1,T):- door(D,C1,C2), not holds_at(door_open(D),T).
invisible_corner(C1,T):- door(D,C2,C1), not holds_at(door_open(D),T).
```

Fig. 7. Sample background information available to the RNC (Taken from (Shanahan, 2000)) including: 1) The robot's initial situation; 2) The robot's narrative of events – a plan of execution; 3) The encoding of the layout of room 2 of the "model office"; 4) Miscellaneous predicates.

```
initiates(follow_wall,loc(corner(C2),ahead),T):-
    holds_at(loc(corner(C1),behind),T),
    next_visible_corner(C1,C2,T).
terminates(follow_wall,loc(corner(C1),behind),T):-
    holds_at(loc(corner(C1),behind),T).
initiates(go_straight,loc(corner(C2),behind),T):-
    holds_at(loc(corner(C1),ahead),T), door(_D,C1,C2).
terminates(go_straight,loc(corner(C1),ahead),T):-
    holds_at(loc(corner(C1),ahead),T),door(_D,C1,_C2).
initiates(go_straight,loc(corner(C2),behind),T):-
    holds_at(loc(corner(C1),ahead),T), door(_D,C2,C1).
terminates(go_straight,loc(corner(C1),ahead),T):-
    holds_at(loc(corner(C1),ahead),T),door(_D,_C2,C1).
initiates(turn(left),loc(corner(C2),behind),T):-
    holds_at(loc(door(D),in),T), holds_at(in(R1),T),
    connects(D,R1,R2), door(D,C1,C2),
    nearest_corner(R2,C1,C2).
terminates(turn(left),loc(corner(C1),ahead),T):-
    holds_at(loc(corner(C1),ahead),T), door(D,C1,_C2),
    holds_at(door_open(D),T).
initiates(turn(right),loc(corner(C2),behind),T):-
    holds_at(loc(door(D),in),T), holds_at(in(R1),T),
    connects(D,R1,R2), door(D,C2,C1), nearest_corner(R2,C2,C1).
terminates(turn(right),loc(corner(C1),ahead),T):-
    holds_at(loc(corner(C1),ahead),T), door(D,_C2,C1),
    holds_at(door_open(D),T).
initiates(turn(left),loc(door(D),in),T):-
    holds_at(loc(corner(C1),ahead),T), door(D,C1,_C2),
    holds_at(door_open(D),T).
terminates(turn(left),loc(door(D),in),T):-
    holds_at(loc(door(D),in),T).
initiates(turn(right),loc(door(D),in),T):-
    holds_at(loc(corner(C1),ahead),T), door(D,_C2,C1),
    holds_at(door_open(D),T).
terminates(turn(right),loc(door(D),in),T):-
    holds_at(loc(door(D),in),T).
initiates(turn(left),in(R2),T):-
    holds_at(loc(door(D),in),T), holds_at(in(R1),T), connects(D,R1,R2).
terminates(turn(left),in(R1),T):-
    holds_at(loc(door(_D),in),T), holds_at(in(R1),T).
initiates(turn(right),in(R2),T):-
    holds_at(loc(door(D),in),T), holds_at(in(R1),T), connects(D,R1,R2).
terminates(turn(right),in(R1),T):-
    holds_at(loc(door(_D),in),T), holds_at(in(R1),T).
```

Fig. 8. The Event Calculus domain specific axioms to describe the Robot Navigation Control program.

Learning Structure and Parameters of Stochastic Logic Programs

Stephen Muggleton

Department of Computing, Imperial College, London, UK

Abstract. Previous papers have studied learning of Stochastic Logic Programs (SLPs) either as a purely parametric estimation problem or separated structure learning and parameter estimation into separate phases. In this paper we consider ways in which both the structure and the parameters of an SLP can be learned simultaneously. The paper assumes an ILP algorithm, such as Progol or FOIL, in which clauses are constructed independently. We derive analytical and numerical methods for efficient computation of the optimal probability parameters for a single clause choice within such a search.

Keywords: Stochastic logic programs, generalisation, analytical methods, numerical methods.

1 Introduction

Stochastic Logic Programs (SLPs) [11] were introduced originally as a way of lifting stochastic grammars to the level of first-order Logic Programs (LPs). Later Cussens [1] showed that SLPs can be used to represent undirected Bayes' nets. SLPs have been used [9] to define distributions for sampling within Inductive Logic Programming (ILP) [7].

Previous papers have studied learning of Stochastic Logic Programs (SLPs) either as a purely parametric estimation problem [2] or separated structure learning and parameter estimation into separate phases [12]. In this paper we consider ways in which both the structure and the parameters of an SLP can be learned simultaneously. We assume an ILP algorithm, such as Progol [6] or FOIL [14], in which clauses are constructed independently. Analytical and numerical methods are derived for efficient computation of the optimal probability label for a single clause choice within such a search.

The paper is arranged as follows. Section 2 gives a general introduction to SLPs. The generalisation model for SLPs described in [12] is reviewed in Section 3. Section 4 describes the problem of choosing the probability label for a single new clause, assuming all other clauses and probability labels in the SLP remain fixed. Equations for the general case are derived using calculus in Section 4.1. A closed form solution for the case in which exactly two examples are present is described in Section 4.2. Iterative numerical methods for solving the general case are described in Section 4.3. Related work is then discussed in Section 5. Section 6 describes provides conclusions and a description of further work.

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 198–206, 2003.

2 Stochastic Logic Programs

2.1 Syntax of SLPs

An SLP S is a set of labelled clauses p:C where p is a probability (ie. a number in the range $[0, 1]$) and C is a first-order range-restricted definite clause[1]. The subset S_p of clauses in S with predicate symbol p in the head is called the definition of p. For each definition S_p the sum of probability labels π_p must be at most 1. S is said to be complete if $\pi_p = 1$ for each p and incomplete otherwise. $P(S)$ represents the definite program consisting of all the clauses in S, with labels removed.

Example 1. **Unbiased coin.** The following SLP is complete and represents a coin which comes up either heads or tails with probability 0.5.

$$S_1 = \left\{ \begin{array}{l} 0.5 : coin(head) \leftarrow \\ 0.5 : coin(tail) \leftarrow \end{array} \right\}$$

S_1 is a simple example of a sampling distribution

Example 2. **Pet example.** The following SLP is incomplete.

$$S_2 = \left\{ \begin{array}{r} 0.3 : likes(X, Y) \leftarrow pet(Y, X), pet(Z, X), \\ cat(Y), mouse(Z) \end{array} \right\}$$

S_2 shows how statements of the form $\Pr(P(\boldsymbol{x})|Q(\boldsymbol{y})) = p$ can be encoded within an SLP, in this case Pr(likes(X,Y)|...)) = 0.3.

2.2 Proof for SLPs

A Stochastic SLD (SSLD) refutation is a sequence $D_{S,G} = \langle 1{:}G, p_1{:}C_1, .., p_n{:}C_n \rangle$ in which G is a goal, each $p_i{:}C_i \in S$ and $D_{P(S),G} = \langle G, C_1, .., C_n \rangle$ is an SLD refutation from $P(S)$. SSLD refutation represents the repeated application of the SSLD inference rule. This takes a goal p:G and a labelled clause q:C and produces the labelled goal pq:R, where R is the SLD resolvent of G and C. The answer probability of $D_{S,G}$ is $Q(D_{S,G}) = \prod_{i=1}^{n} p_i$. The incomplete probability of any ground atom a with respect to S is $Q(a|S) = \sum_{D_{S,(\leftarrow a)}} Q(D_{S,(\leftarrow a)})$. We can state this as $S \vdash_{SSLD} Q(a|S) \leq Pr(a|S) \leq 1$, where $Pr(a|S)$ represents the conditional probability of a given S.

Remark 1. **Incomplete probabilities.** If a is a ground atom with predicate symbol p and the definition S_p in SLP S is incomplete then $Q(a|S) \leq \pi_p$.

Proof. Suppose the probability labels on clauses in S_p are $p_1, .., p_n$ then $Q(a|S) = p_1q_1 + .. + p_nq_n$ where each q_i is a sum of products for which $0 \leq q_i \leq 1$. Thus $Q(a|S) \leq p_1 + .. + p_n = \pi_p$.

[1] Cussens [1] considers a less restricted definition of SLPs.

3 Generalisation Model

We assume an ILP framework in which we are provided with a background SLP S which corresponds to the underlying logic program $B = P(S)$. It is further assumed that S is complete. In addition we are given a set of ground unit positive examples E. The aim is to construct a labelled definite clause $x : H$ which when added to S gives the SLP S'. The clause H must be such that

$$B \wedge H \models E.$$

Suppose $x : H$ is placed within definition S'_q. By replacing all other labels y_i in S_q by $y_i(1-x)$ in S'_q we can ensure that π_q remains 1. The label x is chosen to maximise the likelihood $p(E|S')$ where

$$p(E|S') = \prod_{e \in E} \sum_{p \in SS(e,S')} \prod_{l \text{ in } p} l \tag{1}$$

and $SS(e, S')$ represents the set of SSLD derivations of e from S'.

4 Optimal Parameter Choice

Remark 2. Assuming that S is non-recursive, the term $\prod_{l \in p} l$ either has the form

c *(where c is a constant) if proof p does not involve a q clause,*
cx *if proof p involves $x : H$ or*
c(1-x) *if proof p involves a clause other than $x : H$ in q.*

If we vary the probability label x then $p(E|S')$ is maximal when

$$\frac{\mathrm{d}\ p(E|S')}{dx} = 0 \tag{2}$$

Unfortunately the term-size of a differential of a product of sums of products increases rapidly in the number of summed terms due to the form of the product rule. We take the alternative route of differentiating $\ln p(E|S')$. The following theorem allows us to identify the solution of Equation (2) with the solution for the maximum of $\ln p(E|S')$.

Theorem 1. *Every differentiable function $f(x)$ is maximal when $\frac{d\ ln\ f(x)}{dx} = 0$.*
Proof. *Let $g(x) = \frac{d\ f(x)}{dx}$ and $h(x) = \frac{d\ ln\ f(x)}{dx} = \frac{1}{f(x)} \frac{d\ f(x)}{dx} = \frac{g(x)}{f(x)}$. From calculus $f(x)$ is maximal when $g(x) = 0$. In this case $h(x) = \frac{0}{f(x)} = 0$ when $f(x) \neq 0$.*

4.1 General Case

Given the assurances of Theorem 1 we now log transform Equation (1) as follows.

$$\begin{aligned}
\ln \mathrm{p}(E|S') &= \sum_{e\in E} \ln \left[\sum_{p\in SS(e,S')} \prod_{l\in p} l \right] \\
&= \sum_{e\in E} \ln U(e,x) \\
\frac{\mathrm{d} \ln \mathrm{p}(E|S')}{dx} &= \sum_{e\in E} \frac{\mathrm{d} \ln U(e,x)}{dx} \\
&= \sum_{e\in E} \frac{1}{U(e,x)} \frac{\mathrm{d}\, \mathrm{U}(e,x)}{dx} \qquad (3)
\end{aligned}$$

From Remark 2 above we get the following.

$$U(e,x) = x(c_1 + c_2 + \ldots) + (1-x)(d1 + d2 + \ldots) + c(e) \qquad (4)$$

where c_i, d_j are products of probability labels from S and $c(e)$ is a sum of products of labels from S. We can now simplify Equation (4) as follows.

$$U(e,x) = xk_1(e) + k_2(e) \qquad (5)$$

where $c = c_1 + c_2 \ldots$, $d = d_1 + d_2 \ldots$, $k_1(e) = (c-d)$, $k_2(e) = d + c(e)$. Combining Equation (4) and (5) gives the following.

$$\begin{aligned}
\frac{\mathrm{d} \ln \mathrm{p}(E|S')}{dx} &= \sum_{e\in E} \frac{k_1(e)}{xk_1(e) + k_2(e)} \\
&= \sum_{e\in E} \frac{1}{x + \frac{k_2(e)}{k_1(e)}} \\
&= \sum_{e\in E} \frac{1}{x + k(e)} \qquad (6)
\end{aligned}$$

where $k(e) = \frac{k_2(e)}{k_1(e)}$.

The following theorem defines the general case for finding the value of x which maximises $p(E|S')$.

Theorem 2. *If $p(E|S) \neq 0$ then $p(E|S')$ is maximal when x takes a value defined by $\sum_{e\in E} \frac{1}{x+k(e)} = 0$.*

Proof. Follows trivially from Theorem 1 and Equation (6).

4.2 Analytical Solution for Two Example Case

We now demonstrate how the general equation from Theorem 2 can be solved analytically for the case in which E contains only two examples, e_1 and e_2. In terms of ILP this corresponds to the case of finding the probability label for the least general generalisation of two examples. In this case

$$\sum_{e \in E} \frac{1}{x + k(e)} = 0$$
$$\Rightarrow (x + k(e_1)) + (x + k(e_2)) = 0$$
$$\Rightarrow x = -\left[\frac{k(e_1) + k(e_2)}{2}\right]. \quad (7)$$

We demonstrate this analytical solution below by way of an example.

Example 3. Let S' be

$$S' = \begin{cases} x: & p(X,Y) \leftarrow q(X,Z), r(Z,Y) & [A] \\ 1-x: & p(X,Y) \leftarrow r(X,Z), s(Y,Z) & [B] \\ & & \\ 0.3: & q(a,b) \leftarrow & [C] \\ 0.4: & q(b,b) \leftarrow & [D] \\ 0.3: & q(c,e) \leftarrow & [E] \\ & & \\ 0.4: & r(b,d) \leftarrow & [F] \\ 0.6: & r(e,d) \leftarrow & [G] \\ & & \\ 0.9: & s(d,d) \leftarrow & [H] \\ 0.1: & s(e,d) \leftarrow & [I] \end{cases}$$

and E be

$$E = \begin{cases} p(b,d) \leftarrow [e_1] \\ p(c,d) \leftarrow [e_2] \end{cases}$$

Now the proofs for e_1 and e_2 are as follows.

$$\text{SSLD}(e_1, S) = \{\langle A, D, F\rangle, \langle B, F, H\rangle\}$$
$$\text{SSLD}(e_2, S) = \{\langle A, E, G\rangle, \langle B, G, I\rangle\}$$

Given these proofs the likelihood function is as follows.

$$p(E|S) = [x(0.4)(0.4) + (1-x)(0.4)(0.9)] * [x(0.3)(0.6) + (1-x)(0.6)(0.1)]$$

This function is plotted in Figure 1. Now for e_1 we have the following.

$$U(e_1, x) = x(c_1 + c_2 + \ldots) + (1-x)(d_1 + d_2 + \ldots) + c(e_1)$$
$$c = c_1 = (0.4)(0.4) = 0.16$$
$$d = d_1 = (0.4)(0.9) = 0.36$$
$$c(e_1) = 0$$
$$k_1(e_1) = (c - d) = -0.20$$
$$k_2(e_1) = (d + c(e_1)) = 0.36$$
$$k(e_1) = \frac{k_2(e_1)}{k_1(e_1)} = \frac{0.36}{-0.20} = -1.8$$

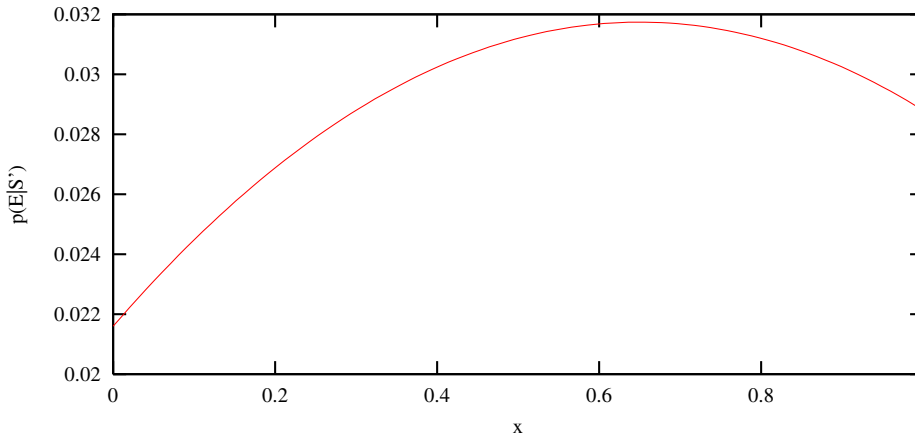


Fig. 1. Likelihood function for Example 3

By similar reasoning we have the following for e_2.

$$k(e_2) = \frac{k_2(e_2)}{k_1(e_2)} = \frac{0.06}{0.12} = 0.5$$

Thus according to Equation (7) the optimal value for x is as follows.

$$\begin{aligned} x &= -\frac{k(e_1) + k(e_2)}{2} \\ &= 0.65 \end{aligned}$$

4.3 Iterative Numerical Method for n Example Case

It is in fact possible to find analytical solutions for numbers of examples $n = 3, 4, \ldots$. This is done by solving quadratic, cubic and higher-order equations. However, the size of the functions to be solved grows exponentially in n, and some of the roots are imaginary. As an alternative to this approach we can use numerical techniques to solve roots of the general equation found in Theorem 2. Standard methods here include Newton's method and the iteration method[2]. In practice the author has found that the following variant of the iteration method appears to converge rapidly.

Variant of Iteration Method. Suppose that you can put an equation g(x)=0 into the form x = f(x). Start with an approximation x_0 of the root. Calculate $x_0, x_1, \ldots, x_n, \ldots$ such that

$$x_{i+1} = \frac{x_i + f(x_i)}{2}$$

[2] See http://www.ping.be/math/root.htm

The sequence converges to the root as long as the initial approximation is sufficiently close. The following is a derivation from the general equation of Theorem 2 of a suitable function $f(x)$.

$$\sum_{e \in E} \frac{1}{x + k(e)} = 0$$

$$\sum_{e \in E \setminus e_1} \frac{1}{x + k(e)} = -\left[\frac{1}{x + k(e1)}\right]$$

$$x = -\left[k(e_1) + \frac{1}{\sum_{e \in E \setminus e_1} \frac{1}{x + k(e)}}\right]$$

$$= f(x)$$

Example 4. We revisit example 3.

$$f(x) = -\left[-1.8 + \frac{1}{\sum_{ein E \setminus e_1} \frac{1}{x + k(e)}}\right]$$

$$= -\left[-1.8 + (x + 0.5)\right]$$

$$= 1.8 - 0.5 - x$$

$$= 1.3 - x$$

Choosing $x_0 = 0.5$ as the initial guess the sequence converges as follows.

$$x_0 = 0.5$$

$$x_1 = 0.65$$

$$x_2 = 0.65$$

5 Discussion of Related Work

This section describes some of the related approaches which have been taken to learning probabilistic logic representations.

5.1 Learning PRMS

PRMs share the underlying probabilistic semantics and local independence assumptions of Bayesian Networks. This has allowed many of the Bayesian net learning techniques to be extended to PRMs. For instance, Koller and Pfeffer [5] have used EM (Expectation Maximisation [3]) to estimate the parameters θ_S of a PRM for which the dependency structure is known. Unlike the algorithms described in the present paper, EM is not guaranteed to converge to optimal values. However, the multi-variate parameter estimation problem being attacked by Koller and Pfeffer using EM is considerably harder than the univariate problem considered here. More recently Friedman et al. [4] have also attacked the more difficult problem of learning the dependency structure S directly from data.

5.2 Learning SLPs

The task of learning SLPs, like that of learning PRMs, has previously been divided into that of parameter estimation and structure learning. Cussens [2] presents an algorithm called Failure-Adjusted Maximisation (FAM) which estimates from data the parameters of an SLP for which the underlying logic program is given. FAM is an instance of the EM algorithm that applies specifically to normalised SLPs. Recently the author [12] presented a two-phase algorithm for learning both the parameters of an SLP and the underlying logic program from data. The algorithm is based on maximising Bayes' posterior probability, and has been demonstrated on problems involving learning an animal taxonomy and a simple English grammar.

6 Conclusions and Further Work

In this paper we have considered ways in which both the structure and the parameters of an SLP can be learned simultaneously. The paper assumes an ILP algorithm, such as Progol or FOIL, in which clauses are constructed independently. We have derive analytical and numerical methods for efficient computation of the optimal probability parameters for a single clause choice within such a search.

Further analysis is required for some of the approaches described in this paper. For instance, the simplifying assumption made in Remark 2 that S be non-recursive may be overly-restrictive in some cases, though it is almost identical to the C-derivation assumption used in relative least generalisation [13,10]. Also the convergence rate of the iteration method described in Section 4.3 needs to be analysed, and compared to alternatives approaches. In particular, empirical comparisons should be made against Cussen's FAM algorithm [2] when the latter is restricted to the case of single label estimation.

Further work is required to implement and test the approach described in this paper. The intention is to incorporate the iterative numerical method described in Section 4.3 within a version of Progol. The performance of this new version of Progol can then be compared with the two stage implementation described in [8]. The author believes that the new approach has the potential to improve on that described in [8] in terms of both efficiency and accuracy of the generated result.

Acknowledgements

Many thanks are due to my wife, Thirza and daughter Clare for the support and happiness they give me. This work was supported partly by the ESPRIT IST project "Application of Probabilistic Inductive Logic Programming (APRIL)", the EPSRC grant "Closed Loop Machine Learning" and the BBSRC/EPSRC Bio-informatics and E-Science Programme, "Studying Biochemical networks using probabilistic knowledge discovery".

References

1. J. Cussens. Loglinear models for first-order probabilistic reasoning. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence*, pages 126–133, San Francisco, 1999. Kaufmann.
2. J. Cussens. Parameter estimation in stochastic logic programs. *Machine Learning*, 2000. In press.
3. A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.
4. N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI-99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1300–1309, San Mateo, CA:, 1999. Morgan-Kaufmann.
5. D. Koller and A. Pfeffer. Learning probabilities for noisy first-order rules. In *IJCAI-97: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1316–1321, San Mateo, CA:, 1997. Morgan-Kaufmann.
6. S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.
7. S. Muggleton. Inductive logic programming: issues, results and the LLL challenge. *Artificial Intelligence*, 114(1–2):283–296, December 1999.
8. S. Muggleton. Learning stochastic logic programs. *Electronic Transactions in Artificial Intelligence*, 5(041), 2000.
9. S. Muggleton. Learning from positive data. *Machine Learning*, 2001. Accepted subject to revision.
10. S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, pages 368–381, Tokyo, 1990. Ohmsha.
11. S.H. Muggleton. Stochastic logic programs. In L. de Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, 1996.
12. S.H. Muggleton. Learning stochastic logic programs. In Lise Getoor and David Jensen, editors, *Proceedings of the AAAI2000 workshop on Learning Statistical Models from Relational Data*. AAAI, 2000.
13. G. Plotkin. A further note on inductive generalization. In *Machine Intelligence*, volume 6. Edinburgh University Press, 1971.
14. J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

A Novel Approach to Machine Discovery: Genetic Programming and Stochastic Grammars

Alain Ratle[1] and Michèle Sebag[2]

[1] LRMA- Institut Supérieur de l'Automobile et des Transports 58027 Nevers France
Alain.Ratle_isat@u-bourgogne.fr
[2] LRI CNRS UMR 8623, Université Paris Sud, 91405 Orsay France
Michele.Sebag@lri.fr

Abstract. The application of Genetic Programming (GP) to the discovery of empirical laws most often suffers from two limitations. The first one is the size of the search space; the second one is the growth of non-coding segments, the introns, which exhausts the memory resources as GP evolution proceeds.
These limitations are addressed by combining Genetic Programming and Stochastic Grammars. On one hand, grammars are used to represent prior knowledge; for instance, context-free grammars can be used to enforce the discovery of dimensionally consistent laws, thereby significantly restricting GP search space. On the other hand, in the spirit of distribution estimation algorithms, the grammar is enriched with derivation probabilities. By exploiting such probabilities, GP avoids the intron phenomenon.
The approach is illustrated on a real-world like problem, the identification of behavioral laws in Mechanics.

1 Introduction

Genetic Programming (GP) extends the principles of Genetic Algorithms and Evolutionary Computation [3] to optimization in tree-structured spaces [8,2]. This paper is more particularly concerned with the use of GP for the automatic discovery of empirical laws. Although GP is widely used for symbolic regression [12,5], it suffers from two main limitations. One first limitation is that canonical GP offers no tractable way to incorporate significant domain knowledge, despite the fact that the knowledge-based issues of Evolutionary Computation have been widely acknowledged [17,7].

Significant advances in this respect have been done by Gruau [6] and Whigham [26], combining context free grammars (CFG) and GP to express and enforce syntactic constraints on the GP solutions. In a previous work [18], grammar-based GP was used to enforce the discovery of dimensionally consistent laws. Indeed, in virtually all physical applications, the domain variables are labeled with their dimensions (units of measurement), and the solution law must be consistent with respect to these units (seconds and meters should not be added). Dimensional consistency constitutes a widely available background knowledge; still it allows for massive contractions of the GP search space.

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 207–222, 2003.

However, grammar-based GP faces specific difficulties when dealing with complex grammars [19]; and dimensional grammar is much more complex than previous grammars used in combination with GP in the literature [6,13,26,15]. An ad-hoc GP initialization procedure, based on attribute-like grammars, was therefore devised [18].

A second limitation of GP is that it requires huge amounts of computational resources, even when the search space is properly constrained. This is blamed on the bloat phenomenon, resulting from the growth of non-coding branches (*introns*) in the GP individuals [8,9]. While its causes are not yet clearly understood, the bloat phenomenon adversely affects GP in two ways; on one hand, it might cause the early termination of the GP runs due to the exhaustion of available memory; on the other hand, it significantly increases the fitness computation cost.

In this paper a novel GP scheme addressing the bloat phenomenon is presented, which combines grammar-based GP and Distribution Estimation Algorithms [1,16,11]. Distribution estimation algorithms maintain a distribution over the search space (as opposed to maintaining a population of individuals, which standard EC does); in each step, a number of individuals is generated from scratch from the current distribution; these individuals then are evaluated according to the optimization goal, and the distribution is biased toward the best individuals.

The coupling of grammar-based GP and Distribution Estimation Algorithms most naturally considers stochastic grammars: the distribution is encoded through probabilities on the grammar derivations. This scheme called stochastic grammar-based GP, significantly differs from related works [20,22,25].

Stochastic grammar-based GP is validated experimentally on problems inspired from real-world applications in Solid Mechanics, the discovery of behavioral laws from experimental data (section 5).

The main result of the paper is that stochastic-grammar based GP does prevent the uncontrolled growth of introns, while efficiently identifying or approximating the target laws. This result, addressing one major limitation of GP, leads to a novel understanding of the bloat phenomenon.

2 Symbolic Regression with Evolutionary Computation

This section summarizes the bases of our framework: Genetic Programming [8,2], grammar-based GP [6,26], how to handle complex grammars within GP [18], and Distribution Estimation Algorithms [1,11].

2.1 Genetic Programming

We assume the reader's familiarity with the principles of Evolutionary Computation [3]. The fitness function $\mathcal{F}$ is defined from the search space Ω onto $\mathbb{R}$, and artificial evolution maintains a set of points of Ω, also called *population*, in order to reach the possibly many optima of $\mathcal{F}$.

$$\mathcal{F} : \Omega \mapsto \mathbb{R}$$

The initial population is most often built by uniformly sampling the search space. Thereafter, at each step or *generation*, the population is modified using selection (biased toward the best fit points or *individuals*), mutation (operator on Ω) and crossover (operator on Ω^2). Evolution thus proceeds until the stop criterion is met.

Genetic Programming, a novel branch in Evolutionary Computation, extends EC to structured search spaces, more particularly tree-structured spaces [8,2]. Ω denotes thereafter the tree-structured space built from the set of nodes $\mathcal{N}$ and the set of leaves or terminals $\mathcal{T}$.

For instance, the set of polynoms of variables X_1, X_2.. can be represented as a tree-structured search space, where the nodes are the addition, subtraction and multiplication operators ($\mathcal{N} = \{+, -, \times\}$), and the leaves are the variables and real-valued constants ($\mathcal{T} = \{X_1, X_2, \ldots \mathcal{R}\}$, where $\mathcal{R}$ stands for any real-valued constant).

If the structure of the target polynom is known before hand, its identification boils down to optimization in $\mathbb{R}^n$ (finding the coefficients most attuned to the task at hand). The particularity of Genetic Programming is to tackle both the identification of the structure, and the coefficients of the target solution.

The individual representation, also known as genotypic search space, is tightly related to the so-called genotypic operators, i.e. initialization, crossover and mutation. GP crossover proceeds by swapping two randomly selected subtrees in the current individuals, or parents; the resulting trees are called offspring. Mutation proceeds by replacing one subtree in the parent, with one randomly generated subtree.

2.2 Grammar-Based GP

A context free grammar describes the admissible constructs of a language by a 4-tuple $\{S, N, T, P\}$, where S is the start symbol, N the set of non-terminal symbols, T the set of terminal symbols, and P the production rules. Any expression is iteratively built up from the start symbol by rewriting non-terminal symbols into one of their derivations, as given by the production rules, until the expression contains terminals only. Table 1 shows the CFG describing the polynoms of variables X_1 and X_2, with $\mathcal{R}$ again standing for any valued constant. Grammars enable the expression of problem-specific constraints on the GP search space. These constraints are enforced as follows. On one hand, the recursive application of derivation rules can be characterized as a tree, known

Table 1. Context Free Grammar for polynoms of variables X_1 and X_2

$$
\begin{array}{lll}
N = & \{< E >,\ < Op >,\ < V >\} & \textit{expression, operator, variable} \\
T = & \{+,\ -,\ \times,\ X_1,\ X_2,\ \mathcal{R}\ \} & \\
P = & \left\{ \begin{array}{l} S \ := \ < E > \ ; \\ < E > \ := \ < Op > \ < E > \ < E > \ \mid < V > \ ; \\ < Op > := \ + \ \mid \ - \mid \ \times \ ; \\ < V > \ := \ X_1 \mid X_2 \mid \mathcal{R} \ ; \end{array} \right. &
\end{array}
$$

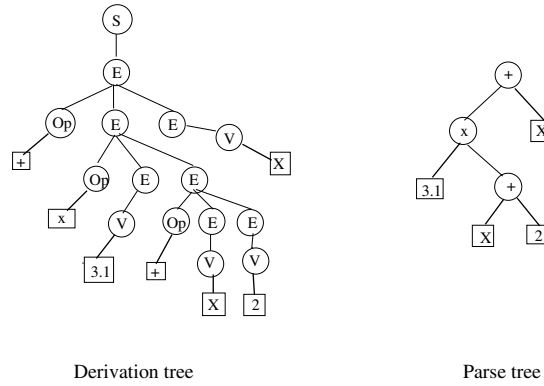


Fig. 1. Derivation tree and corresponding parse tree

as *derivation tree* (Fig. 1); the derivation trees are equivalent to the expression trees, or *parse trees*.

Grammar-based GP manipulates derivation trees instead of parse trees. Crossover is restricted to swapping subtrees built on the same non-terminal symbol; mutation replaces a subtree by a new tree built on the same symbol [6,26]. This ensures that CFG-compliant offspring are produced from CFG-compliant parents. These restrictions are quite similar to that of Strongly Typed GP [13].

2.3 Initialization in Grammar-Based GP

Our initial motivation for grammar-based GP is to enforce the discovery of laws dimensionally consistent wrt the application units, thereby both restricting the search space and increasing the expert's satisfaction.

We restricted ourselves to a finite number of compound units expressed wrt the elementary units; for instance the *Newton* unit is represented as vector $(1, 1, -2)$ wrt to elementary units (*kg, meter and second*). To each allowed unit is associated a non-terminal symbol. The corresponding production rule describes all ways of producing an expression of this type, e.g. a Newton-typed expression is obtained by adding two Newton-typed expressions, by multiplying a mass-typed expression with an acceleration-typed expression, and so forth. This grammar is automatically generated from the application units, and the range of allowed compound units; it describes all dimensionally consistent laws in the search space[1]. Although the CFG size is exponential, enforcing these restrictions linearly increases the crossover complexity in the worst case, and does not modify the mutation complexity.

The dimensional grammar involves more than one hundred non-terminal symbols, and several thousands of derivations, which is considerably more than

[1] The production rule associated to the start symbol specifies the unit of the sought solution; it can also enforce the shape of the solution, according to the expert guess.

the grammars previously considered in the literature [6,26,15]. A severe limitation of grammar-based GP is then noticed [19]. The size of the GP trees normally is limited by setting a maximal tree depth D_{Max}. But expressions generated from the dimensional grammar through uniformly selecting derivation rules are *very* long (each non-terminal symbol is associated with many derivations, few of which are terminal). On one hand, these expressions are massively rejected because of their size, making the initialization step heavily computationally expensive. On the other hand, the expressions finally found in the initial population are poorly diversified, and evolution hardly ever recovers from a poor initial population [4].

A new initialization procedure was therefore designed for grammar-based GP [18], inspired from attribute-grammar. To each non-terminal symbol N is associated an integer value $\ell(N)$, giving the minimal depth of a parse tree rewriting this symbol. An integer value $\ell(d)$ is similarly associated to each derivation. These integers are recursively computed; $\ell(N)$ is set to the minimal $\ell(d)$ over all derivations d used to rewrite N; and $\ell(d)$ is computed from the $\ell(N)$s, over all symbols N occurring in the derivation.

If the minimal tree depth in the language is less than the maximal tree-depth allowed ($\ell(S) \leq D_{Max}$), then by recurrence for all non-terminal symbols in the current expression there exist admissible derivations, i.e. such that they lead to a tree within the tree-depth bound. By uniformly selecting one derivation among the admissible ones, parse trees complying with the grammar and the maximal tree-depth allowed, are produced. As the initial population so built is sufficiently diverse, evolution can work properly [18], until the intron growth is observed.

2.4 Distribution-Based Evolution

Contrasting with genetic evolution, distribution-based evolution deals with a high-level (intentional) description of the best individuals encountered so far, as opposed to the (extensional) description given by the current population itself. This intentional description is a probability distribution $\mathcal{M}$ on the solution space. In each generation, the population is generated from $\mathcal{M}$ only, and the individuals are evaluated; thereafter, $\mathcal{M}$ is biased toward the best individual(s) in the current population, according to a given update mechanism.

As far as we know, the first algorithm resorting to distribution-based evolution is *Population-based Incremental Learning (PBIL)* [1], concerned with optimization in $\{0,1\}^n$. In this scheme, distribution $\mathcal{M}$ is represented as an element of $[0,1]^n$, initialized to $\mathcal{M}_0 = (.5, \ldots, .5)$.

At generation t, $\mathcal{M}_t$ is used to generate the population, the probability for any individual X to have its i-th bit set to 1 being the i-th coordinate of $\mathcal{M}_t$. The best individual X_{best} in the current population is used to update $\mathcal{M}_t$ by relaxation[2], with

$$\mathcal{M}_{t+1} = (1-\epsilon)\mathcal{M}_t + \epsilon X_{best}$$

$\mathcal{M}_t$ is also randomly perturbed (mutated) to avoid premature convergence.

[2] Other variants use the best two individuals, and possibly the worst one too, to update the distribution.

This scheme has been extended to accommodate different distribution models and non-binary search spaces (see [21,11] among others).

3 Stochastic Grammar-Based GP (SG-GP)

This section describes the extension of Distribution Estimation Algorithms to grammar-based GP, which involves the representation of the distribution, its exploitation in order to generate the current population, and its update from the current best (and worst) individuals.

3.1 Overview

Representation of the Distribution. The distribution for grammar-based GP is most naturally represented as a stochastic grammar: each derivation rule d_i in the grammar is enriched with a weight w_i ($w_i \in \mathbb{R}^+$), and the selection probability of d_i depends on w_i as detailed below.

Generation of the Population. The second component is how to exploit the current distribution in order to generate the population in each generation. It must be noted that this step is much similar to the initialization procedure in standard evolution. We accordingly modify the grammar-based initialization procedure (section 2.3), to accommodate for the derivation weights. Practically, all admissible derivations for the current non terminal symbol N are determined from the maximal tree-depth allowed and the position of N in the parse tree. One derivation among the admissible ones is selected according to their (normalized) weights, using a standard roulette mechanism. More formally, the probability p_i of selecting derivation d_i is defined as:

$$p_i = \begin{cases} \dfrac{w_i}{\sum_{k \in \text{admissible derivs.}} w_k} & \text{if } d_i \text{ is an admissible derivation} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Updating the Distribution. After all individuals in the current population have been evaluated, the probability distribution is updated from the N_b best and N_w worst individuals according to the following mechanism: for each derivation d_i,

- Let b denotes the number of individuals among the N_b best individuals that carry derivation d_i; weight w_i is multiplied by $(1+\epsilon)^b$;
- Let w denotes the number of individuals among the N_w worst individuals that carry derivation d_i; weight w_i is divided by $(1+\epsilon)^w$;
- Last, weight w_i is mutated with probability p_m; the mutation either multiplies or divides w_i by factor $(1+\epsilon_m)$.

All w_i are initialized to 1. Note that it does not make sense to have them normalized; they must be locally renormalized before use, depending on the current set of admissible derivations.

Table 2. Parameters of Stochastic Grammar-based Genetic Programming

Parameter	Definition
Parameters specific to SG-GP	
N_b	Number of best individuals for probability update
N_w	Number of worst individuals for probability update
ϵ	Learning rate
p_m	Probability of mutation
ϵ_m	Amplitude of mutation
Canonical GP parameters	
P	Population size
G	Maximum number of generations
D_{max}	Maximum derivation depth

In summary, stochastic grammar based GP involves five parameters besides the three standard GP parameters (Table 2).

3.2 Scalar and Vectorial SG-GP

In the above scheme, the genetic pool is represented by a vector $\mathcal{W}$, coding all derivation weights for all production rules. $\mathcal{W}$ belongs to $\mathbb{R}^{|d|}$, if $|d|$ denotes the total number of derivations in the grammar.

This way, the storage of a variable length population is replaced by the storage of a single fixed size vector; this is in sharp contrast with canonical GP, and more generally, with all evolutionary schemes dealing with variable size individuals.

One side effect of this representation is that it induces a total order on the derivations in a given production rule. However, it might happen that derivation d_i is more appropriate than d_j in upper levels of the GP trees, whereas d_j is more appropriate at the bottom of the trees.

To address this limitation, to each level of the GP trees (i ranging from 1 to D_{max}) is attached a distribution vector $\mathcal{W}_i$. This latter scheme is referred to as *Vectorial SG-GP*, as opposed to the former scheme of *Scalar SG-GP*.

The distribution update in *Vectorial SG-GP* is modified in a straightforward manner; the update of distribution $\mathcal{W}_i$ is only based on the derivations actually occurring at the i-th level among the best and worst individuals in the current population.

4 Experiment Goal

Besides the finding accurate laws, our experiment goal is to gain some insights into the intron growth phenomenon.

It has often been observed experimentally that the proportion of introns in the GP material grows exponentially along evolution [14]. As already mentioned, the intron growth is undesirable as it drains out the memory resources, and increases the total fitness computation cost.

However, it was also observed that pruning the introns in each generation significantly decreases the overall GP performances [8]. Supposedly, introns protect good building blocks from the destructive effects of crossover; as the useful part of the genome is condensed into a small part of the individual, the probability for a crossover to break down useful sections is reduced by the apparition of introns.

Intron growth might also be explained from the structure of the search space [9]. Consider all genotypes (GP trees) coding a given phenotype (program). There exists a lower bound on the genotype size (the size of the shortest tree coding the program); but there exists no upper bound on the genotype size (a long genotype can always be made longer by the addition of introns). Since there are many more long genotypes than short ones, longer genotypes will be selected more often than shorter genotypes (everything else being equal, i.e. assuming that the genotypes are equally fit)[3].

Last, intron growth might also be a mechanical effect of evolution. GP crossover facilitates the production of larger and larger trees: on one hand, the offspring average size is equal to the parent average size; on the other hand, short size offspring usually are poorly fit; these remarks together explain why the individual size increases along evolution.

Since the information transmission in SG-GP radically differs from that in GP, as there exists no crossover in SG-GP, there should be no occasion for building longer individuals, and no necessity for protecting the individuals against destructive crossover.

Experiments with SG-GP are intended to assess the utility of introns. If the intron growth is beneficial *per se*, then either SG-GP will show able to produce introns — or the overall evolution results will be significantly degraded. If none of these eventualities is observed, this will suggest that the benefits of introns have been overestimated.

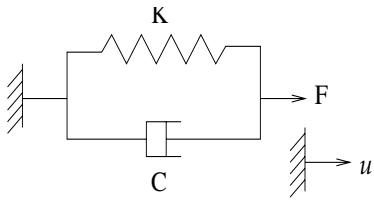
5 Experimental Validation

This section presents the experimental validation of the SG-GP scheme on problems inspired from real-world applications, related to the identification of rheological models. These problems have important applications in the development of new materials, especially for polymers and composite materials [24].

5.1 Test Problem

The target empirical law corresponds to the Kelvin-Voigt model, which consists of a spring and a dashpot in parallel. When a constant force is applied, the response (displacement-time relation) follows the equation below is given below.

[3] To prevent or reduce this intron growth, a parsimony pressure (regularization term) might be added to the fitness function [27]. In other words, the machine discovery objective is turned into a multi-objective optimization problem, finding an accurate law, and finding a simple law. As widely acknowledged, adjusting the relative weights of the two objectives is far from easy.



$$u(t) = \frac{F}{K}\left(1 - e^{\frac{-Kt}{C}}\right)$$

Fig. 2. The Kelvin-Voigt Model

Table 3. Physical Units in the Kelvin-Voigt Model

Physical units			
Quantity	mass	length	time
Variables			
F (Force)	+1	+1	–2
K (Elastic elements)	+1	0	–2
C (Viscous elements)	+1	0	–1
t (time)	0	0	+1
Solution			
u (displacement)	0	+1	0

Fitness cases (examples) are generated using random values of the material parameters K and C and loading F. The physical units for the domain variables and for the solution are expressed with respect to the elementary *mass*, *time* and *length* units (Table 3). Compound units are restricted as the exponent for each elementary unit ranges in $\{-2, -1, 0, 1, 2\}$. The dimensional grammar is generated as described in section 2.3, with 125 non-terminal symbols and four operators (addition, multiplication, protected division and exponentiation). The grammar size is about 515 k.

5.2 Experimental Setting

SG-GP is compared with standard elitist GP. The efficiency of SG-GP is assessed with respect to the quality of solutions, and in terms of memory use. All results are averaged on 20 independent runs.

GP and SG-GP parameters are set according to a few preliminary experiments (Table 4). Canonical GP is known to work better with large populations and small number of generations [8]. Quite the contrary, SG-GP works better with a small population size and many generations. In both cases, evolution is stopped after 2,000,000 fitness evaluations.

5.3 Experimental Results

Fig. 3 shows the comparative behaviors of canonical GP and scalar stochastic grammar-based GP on the test identification problems. The performance measures the average best fitness (mean squared distance over all 20 time steps, and 20 mechanical experiments), over 20 independent runs. The performances

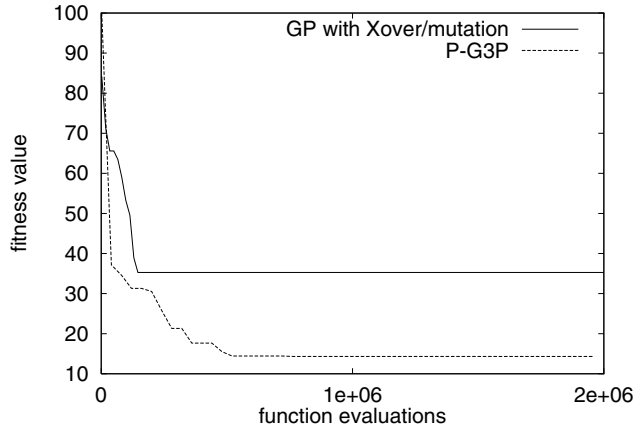


Fig. 3. Comparative performances of GP and Scalar stochastic grammar based GP (averaged on 20 runs)

Table 4. Optimization parameters

SG-GP		GP	
Parameter	Value	Parameter	Value
Population size	500	Population size	2000
Max. number of generations	4000	Max. number of generations	1000
N_b	2	P(crossover)	0.9
N_w	2	P(mutation)	0.5
ϵ	0.001	Tournament size	3
P_m	0.001		
ϵ_m	0.01		

Table 5. Exact Identification of the target solution by Scalar and Vectorial SG-GP

	Scalar	Vectorial
Number of hits	0 / 20	5 / 20
Av. best fitness	16.7 ± 5	10.3 ± 5

are significantly improved by using Vectorial SG-GP instead of Scalar SG-GP (Fig. 4). More precisely, Vectorial SG-GP finds the target law (up to algebraic simplifications) after 2,000,000 fitness evaluations on 5 out of 20 runs, while no perfect match could be obtained with scalar SG-GP (Table 5).

Grammar-based GP results, not shown here for space limitations, demonstrate that even scalar SG-GP is more efficient than grammar-based GP (see [18] for more details).

5.4 Sensitivity Analysis

The sensitivity analysis studies the influence of the learning rate ϵ and the mutation amplitude ϵ_m on the overall performances of SG-GP (Fig. 5). Overall,

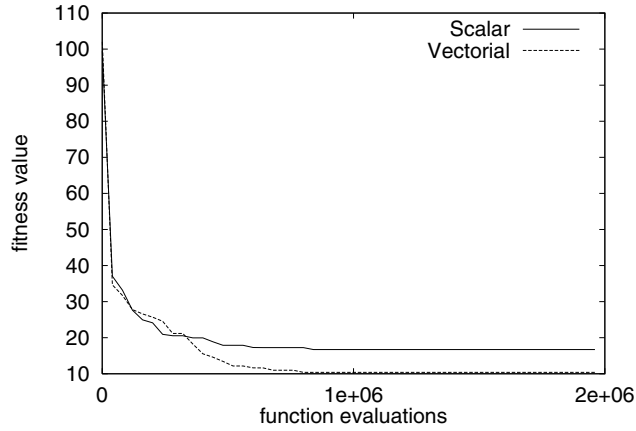
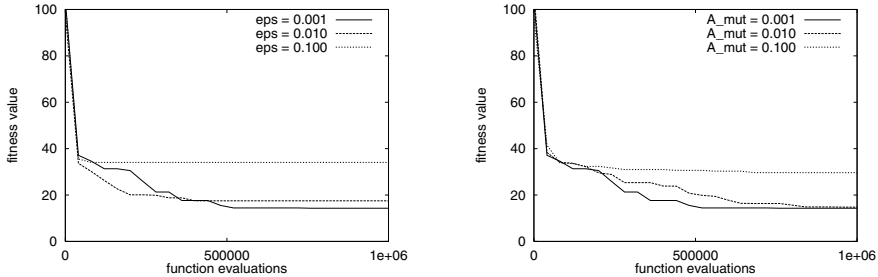


Fig. 4. Comparative performances of Scalar and Vectorial SG-GP (averaged on 20 runs)



(a) Learning rate (b) Mutation amplitude

Fig. 5. Parametric study of Stochastic Grammar based GP

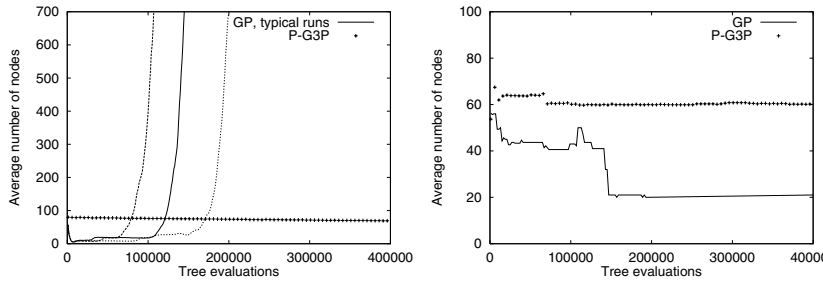
better results are obtained with a low learning rate and a sufficiently large mutation amplitude; this can be interpreted as a pressure toward the preservation of diversity.

The maximum derivation depth allowed D_{max} is also a critical parameter. Too short, and the solution will be missed, too large, the search will take a prohibitively long time.

5.5 Resisting the Bloat

The most important experimental result is that SG-GP *does resist* the bloat, as it maintains an almost constant number of nodes. The average results over all individuals and 20 runs is depicted on Fig. 6.a.

In comparison is shown the number of nodes in GP (averaged on all individuals, but plotted for three typical runs for the sake of clarity). The individual size first drops in the first few generations; and after a while, it suddenly rises exponentially until the end of the run. The drop is due to the fact that many



(a) Average individual size (b) Best individual size

Fig. 6. Solution size with GP and SG-GP, averaged on 20 runs

trees created by crossover in the first generations are either trivial solutions (very simple trees) or infeasible solutions which are rejected. The rise occurs as soon as large feasible trees emerge in the population.

As noted by [2], the size of the best individual is not correlated with the average size. Figure 6.b shows the average size of the best-so-far individual. Interestingly, SG-GP maintains an almost constant size for the best individual, which is slightly less than the average size. On the opposite, GP converges to a very small solution, despite the fact that most solutions are very large.

5.6 Identification and Generalization

As already mentioned, SG-GP found the target law in 5 out of 20 runs (up to algebraic simplifications). In most other runs, SG-GP converges toward a local optimum, a simplified expression of which is:

$$u(t) = \frac{2 * t^2 * \frac{F}{K}}{\frac{C}{K} * \frac{C}{K} + 2 * t^2} \tag{2}$$

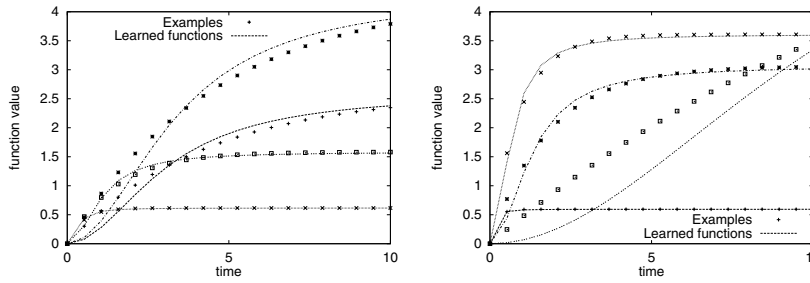
This law is not on the path to the global optimum since the exponential is missing. However, the law closely fits the training examples, at least from an engineering point of view (Fig. 7.a).

Even more important is the fact that SG-GP finds solutions which behaves well on test examples, i.e. examples generated after the target law, which have not been considered during evolution.

By construction, the target law perfectly fits the test examples. But the non-optimal law (Eq 2) also fits the test examples; the fit is quite perfect is three out of four cases, and quite acceptable, again from an engineer's point of view, in the last case (Fig. 7.b).

6 Related Works

Since the pioneering work by Langley et al. [10], many approaches have been proposed for Machine Discovery (see among others [23,25,22]). Rooted in the



(a) On training examples (b) On test examples

Fig. 7. Correct Identification and Generalization with SG-GP

Machine Learning framework, these approaches rely on restricted assumptions (the data generator being a controlled process) [10]; or they demand the expert strong support, either interactively [23] or through background knowledge [22,25].

The cited approaches and more generally, the Machine Discovery approaches develop clever heuristics. However, these heuristics either assume that exhaustive exploration is feasible, or that the target solution can be found through a greedy optimization process.

Schematically, stochastic-grammars based GP should be viewed as complementary to Machine Discovery; as usual, deterministic heuristics are much more efficient than stochastic ones for small or medium size problems. Still, their use sets strong requirements on the size of the search space, or equivalently on the available background knowledge. This contrasts with the wide usability of EC; even more so as SG-GP offers ways of incorporating powerful, still widely available background knowledge.

Our approach is also related to the *Probabilistic Incremental Program Evolution (PIPE)* system [20], which is also concerned with extending distribution-based evolution to GP. In PIPE, the distribution on the GP search space is represented as a Probabilistic Prototype Tree (PPT); each PPT node stores the probabilities for selecting any variable and operator in this node. One main difference is that the PPT size grows exponentially with the average tree depth, whereas the distribution size remains linear in SG-GP. This is visible as PIPE does not avoid the bloat phenomenon, contrasting with SG-GP.

7 Conclusion

In this paper was presented a novel Genetic Programming scheme, combining grammar-based GP [6,26,18] and distribution-based evolution [1], termed SG-GP for Stochastic Grammar-based Genetic Programming. SG-GP shows good identification abilities on the problem considered, as the target law was discovered in 5 out of 20 runs, while it was never discovered by canonical GP. What is equally important, SG-GP shows good generalization abilities; even in the

cases where the target law was missed, the solutions found by SG-GP have good predictive accuracy on test experimental data. Last, SG-GP successfully completed these tasks by exploring individuals with constant size. No intron growth was observed; the overall memory requirements were lower by several orders of magnitude, than for canonical GP.

Fundamentally, SG-GP remedies two main limitations of GP. On one hand, it allows for exploiting powerful background knowledge, such as dimensional consistency. On the other hand, it successfully resists the bloat phenomenon, avoiding the intron growth.

The latter fact is surprising, for intron growth was considered to be unavoidable for program induction methods with fitness-based selection [9]. This conjecture is infirmed by SG-GP results on a real-world like problem, suggesting that intron growth is not necessary to achieve efficient non parametric learning in a fitness-based context, but might rather be a side effect of crossover-based evolution.

On-going work is concerned with examining the actual limitations of SG-GP through more intensive experimental validation.

One such limitation to be addressed regards the problem of adjusting the constants of the target laws (the parametric optimization problem within the non-parametric one).

In the long run, a key aspect for Machine Discovery with SG-GP is to determine an adequate grammar, i.e. to find prior restrictions on the search space. A perspective for further research is to use grammatical inference from the body of knowledge in the application domain (e.g. laws reported in textbooks) to this aim.

Acknowledgments

The authors are partially supported by ILPNet2 Network of Excellence (INCO 977102).

References

1. S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithms. In A. Prieditis and S. Russel, eds, *Proc. of the 12th Int Conf. on Machine Learning*, pages 38–46. Morgan Kaufmann, 1995.
2. W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. *Genetic Programming — An Introduction On the Automatic Evolution of Computer Programs and Its Applications.* Morgan Kaufmann, 1998.
3. T. Bäck. *Evolutionary Algorithms in theory and practice.* New-York:Oxford University Press, 1995.
4. J.M. Daida. Challenges with verification, repeatability, and meaningful comparison in genetic programming: Gibson's magic. In *Proc. of the Genetic and Evolutionary Conf. 99*, pages 1069–1076. Morgan Kaufmann, 1999.
5. J. Duffy and J. Engle-Warnick. Using symbolic regression to infer strategies from experimental data. In *Evolutionary Computation in Economics and Finance.* Springer Verlag, 1999.

6. F. Gruau. On using syntactic constraints with genetic programming. In P.J. Angeline and K.E. Kinnear Jr., eds, *Advances in Genetic Programming II*, pages 377–394. MIT Press, 1996.
7. C. Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13:189–228, 1993.
8. J. R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Evolution.* MIT Press, Massachusetts, 1992.
9. W. B. Langdon and R. Poli. Fitness causes bloat. In *Soft Computing in Engineering Design and Manufacturing*, pages 13–22. Springer Verlag, 1997.
10. P. Langley, H.A. Simon, and G.L. Bradshaw. Rediscovering chemistry with the BACON system. In R.S Michalski, J.G. Carbonell, and T.M. Mitchell, eds, *Machine Learning: an artificial intelligence approach*, volume 1. Morgan Kaufmann, 1983.
11. P. Larranaga and J. A. Lozano. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, 2001.
12. B. McKay, M.J. Willis, and G.W. Barton. Using a tree structures genetic algorithm to perform symbolic regression. In *IEEE Conf. publications, n. 414*, pages 487–492, 1995.
13. David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.
14. P. Nordin, W. Banzhaf, and F.D. Francone. Introns in nature and in simulated structure evolution. In D. Lundh, B. Olsson, and A. Narayanan, eds, *Biocomputing and Emergent Computation*, pages 22–35. World Scientific, 1997.
15. M. O'Neill and C. Ryan, eds. *Grammatical Evolution.* Genetic and Evolutionary Conf. Workshop, GECCO 2002, 2002.
16. M. Pelikan, D.E. Goldberg and E. Cantu-Paz. BOA: the bayesian optimization algorithm. In *Genetic and Evolutionary Conf., GECCO 1999*, pages 525–532. Morgan Kaufmann, 1999.
17. N. J. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5:183–20, 1991.
18. A. Ratle and M. Sebag. Genetic programming and domain knowledge: Beyond the limitations of grammar-guided machine discovery. In M. Schoenauer et al., editor, *Proc. of the* 6^{th} *Conf. on Parallel Problems Solving from Nature*, pages 211–220. Springer-Verlag, LNCS 1917, 2000.
19. C. Ryan, J.J. Collins, and M. O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In W. Banzhaf, R. Poli, M. Schoenauer, and T.C. Fogarty, eds, *Genetic Programming, First Eur. Workshop, EuroGP98*, pages 83–96. Springer Verlag LNCS 1391, 1998.
20. R. Salustowicz and J. Schmidhuber. Evolving structured programs with hierarchical instructions and skip nodes. In J. Shavlik, editor, *Proc. of the* 15^{th} *Int Conf. on Machine Learning*, pages 488–496. Morgan Kaufmann, 1998.
21. M. Sebag and A. Ducoulombier. Extending population-based incremental learning to continuous search spaces. In Th. Bäck, G. Eiben, M. Schoenauer, and H.-P. Schwefel, eds, *Proc. of the* 5^{th} *Conf. on Parallel Problems Solving from Nature*, pages 418–427. Springer Verlag, 1998.
22. L. Todorovski and S. Dzeroski. Using domain knowledge on population dynamics modeling for equation discovery. In P. Flach and L. De Raedt, eds, *Proc. of the* 12^{th} *Eur. Conf. on Machine Learning*, pages 478–490. Springer Verlag LNAI 2167, 2001.
23. R. Valdes-Perez. Machine discovery in chemistry: New results. *Artificial Intelligence*, 4:191–201, 1995.

24. I.M. Ward. *Mechanical Properties of Solid Polymers.* Wiley, Chichester, 1985.
25. T. Washio, H. Motoda, and Y. Niwa. Discovering admissible simultaneous equation models from observed data. In L. de Raedt and P. Flach, eds, *Proc. of the 12th Eur. Conf. on Machine Learning*, pages 539–551. Springer Verlag LNAI 2167, 2001.
26. P.A. Whigham. Inductive bias and genetic programming. In *IEEE Conf. publications, n. 414*, pages 461–466, 1995.
27. Byoung-Tak Zhang and Heinz Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.

Revision of First-Order Bayesian Classifiers

Kate Revoredo and Gerson Zaverucha

Programa de Engenharia de Sistemas e Computação–COPPE
Universidade Federal do Rio de Janeiro
Caixa Postal 68511, 21945-970, Rio de Janeiro, RJ, Brasil
{kate,gerson}@cos.ufrj.br

Abstract. New representation languages that integrate first order logic with Bayesian networks have been proposed in the literature. Probabilistic Relational models (PRM) and Bayesian Logic Programs (BLP) are examples. Algorithms to learn both the qualitative and the quantitative components of these languages have been developed. Recently, we have developed an algorithm to revise a BLP. In this paper, we discuss the relationship among these approaches, extend our revision algorithm to return the highest probabilistic scoring BLP and argue that for a classification task our approach, which uses techniques of theory revision and so searches a smaller hypotheses space, can be a more adequate choice.

1 Introduction

There are several approaches in the literature for extending Bayesian networks to first-order: probabilistic logic programs [23], probabilistic relational models [15], relational Bayesian nets [9], independent choice logic [25] [26] [27], first-order Bayesian reasoning [5], Bayesian Logic Programs (BLP) [10][11] and Stochastic logic programs [20]. Many of these techniques employ the notion of Knowledge-based Model Construction (KBMC) [7], where first-order rules with associated uncertainty parameters are used as a basis for generating Bayesian networks for particular queries. In BLP, which was successfully compared to all the aforementioned approaches, the first-order rules are represented as a logic program (like Prolog) and, as for Bayesian networks, there is a CPD associated with each Bayesian definite clause.

Algorithm to learn both the qualitative and the quantitative components of a PRM and a BLP were proposed in [6] and [13], respectively. These algorithms evaluate a set of possible structures with a scoring function choosing the one with the highest score.

Both algorithm starts with a initial structure and propose modifications on the entire theory, choosing afterwards the one with the highest score. For a classification task, if the initial theory is approximately correct, it would be more efficient to propose modifications only in those places that are responsible for the misclassifications, analyzing these possible revisions and choosing the one with the highest probabilistic score. Therefore, theory revision [31] would be a good choice, since it identifies the places to be repaired using the training examples

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 223–237, 2003.

and modifies only those. The theory found has the best probabilistic score for the search space aforementioned and should have good classification accuracy. If one is interested and have the time, a maximization procedure, which searches for the highest probabilistic score theory in the entire search space (restricted to the theories consistent with the training data), can be performed. Revision of Bayesian Logic Programs (RBLP), [29], is an algorithm which uses these ideas to revise a BLP.

In this paper, we compare PRM and BLP learning algorithms showing the relationship between them and RBLP, and extend RBLP by presenting the maximization procedure aforementioned.

The paper is laid out as follows. In section 2 we review some background knowledge, in section 3 we present our revision algorithm RBLP and its maximization procedure. The comparison between PRM, BLP learning algorithm and RBLP is done in section 4. Finally, in section 5 we give some conclusions and point out some future work.

2 Background Knowledge

In [29] we proposed the algorithm RBLP to revise a Bayesian logic program using techniques of theory revision and for that we based it in the FORTE algorithm [30]. In this section, we briefly review BLP, FORTE, and the BLP and PRM learning algorithms.

2.1 Bayesian Logic Program

The main difference between Bayesian and classical clauses is that Bayesian atoms represent classes of similar random variables. That is, each ground atom in a BLP represents a random variable. Each random variable can take on various possible values from the (finite) domain D_q of the corresponding Bayesian predicate q. In any state of the world a random variable takes exactly one value. Therefore, a logic predicate q is a special case of a Bayesian one with $D_q = \{true, false\}$.

A Bayesian definite clause is of the form:

$$A|A_1, ..., A_n$$

where $A, A_1, ..., A_n$ are Bayesian atoms and all variables are implicit universal quantified (similarly to Prolog). An example is: $burglary(X)|neighborhood(X)$, where the domains are $D_{burglary} = \{true, false\}$ and $D_{neighborhood} = \{bad, average, good\}$. Roughly speaking a Bayesian definite clause $A|A_1, ..., A_n$ specifies that for each substitution β [19] that grounds the clause, the random variable $A\beta$ depends on $A_1\beta, ..., A_n\beta$.

Similarly to Bayesian networks, for each Bayesian definite clause there is a CPD associated with it, as an example, table 1 shows the CPD associated to the clause $burglary(X)|neighborhood(X)$.

Table 1. CPD associated with the clause $burglary(X)|neighborhood(X)$

neighborhood(X)	burglary(X)	burglary(X)
	true	false
bad	0.6	0.4
average	0.4	0.6
good	0.3	0.7

Since a Bayesian predicate is defined by a set of definite Bayesian clauses, so-called combining rules (as noisy-or) are used to obtain the combining probability distribution of those clauses from the CPDs of each one (they are decomposable).

BLP query-answering procedure employs the notion of Knowledge-based Model Construction (KBMC)[7], where first-order rules with associated uncertainty parameters are used as a basis for generating Bayesian networks for particular queries (examples) [10], [11], [12]. Firstly, the Bayesian network for each example is built and then a Bayesian network inference algorithm is applied to it in order to answer the query.

2.2 FORTE

While in the original Inductive Logic Programming (ILP) a (first-order) logic program (LP) is learned from examples and a domain theory (BK) by just adding new clauses to BK, in Theory Revision the approximately correct domain theory is modified using the examples: it cannot only create new clauses (as in ILP), but also modify and/or delete existing ones.

FORTE[30] is a theory revision system successfully compared in the literature [2]. It is a hill-climbing iterative algorithm, where in each iteration tries to improve the theory's accuracy by identifying points that are responsible for the misclassifications and need to be repaired, called revision points. The top-level algorithm is shown in figure 1.

For each iteration and each example the algorithm annotates the revision points, which can be of generalization or specialization. Each revision point has a potential, which is the number of examples correctly classified which could result from a revision of that point. After finding the revision points, operators for specialization and/or generalization are used accordingly in an attempt to make the theory consistent. Then, each one will receive a score, which is the actual increase in theory accuracy it achieves. In other words, the score of a revision point is the number of examples, that after the proposed revision had been applied, turned to be correctly classified. For each iteration the operator that provides the greater score will be the one implemented.

When analyzing a possible revision we need only to use the subset of training examples consisting of those instances whose proofs rely on the target clause.

2.3 Structure Learning Algorithms

In this section we briefly review the BLP and the PRM structure learning algorithms.

```
repeat
   generate revision points
   sort revision points by potential (high to low)
   for each  revision point
        generate revisions
        update best revision found
   until potential of next revision point is less than the score of the best revision to
                                                                                  date
   if best revision improves the theory
        implement best revision
   end if
until no revision improves the theory
```

Fig. 1. FORTE's top-level refinement algorithm

PRM Learning Algorithm. In [6] an algorithm to automatically learn the structure of a PRM was proposed. This algorithm generates possible acyclic structures and with a scoring function and a search algorithm verifies which one is the best.

The learning task for a Bayesian network is NP-Hard [3]. As PRM learning is at least as hard as Bayesian network learning, an efficient procedure that always finds the highest scoring structure, can not be found. Thus, a hill-climbing search using the scoring function as a metric is performed. The current candidate structure is maintained and iteratively improved. At each iteration, a set of simple local modifications (addition and/or deletion) in the set of parents of the attributes are considered. After scoring all of them, the one with the highest scoring is chosen. The scoring function can be decomposed in local components. This way the search algorithm considers just the component of the scoring function related to the attribute $X.A$ when this is the one being modified. As there are many possibilities of sets of parents and the computation of the scoring function component for each one is expensive, the algorithm at each phase (k) uses sets of potential parents ($Pot_k(X.A)$) and the local modifications are performed considering just these sets.

The advantage of this approach is that it reduces the space of search and the most expensive computations are pre-computed (the joins and the aggregation required in the definition of the parents). This construction, together with the decomposability of the scoring function, allows the steps of the search to be done very efficiently. The success of this approach depends on the choice of the potential parents. Following [4], which examines a similar approach in the context of learning Bayesian networks, an iterative approach was proposed that starts with some structure (possibly one where each attribute does not have any parents) and selects the sets $Pot_k(X.A)$ based on this structure. The search procedure is applied and a new higher scoring structure is chosen. New potential parents are chosen based on this new structure and reiterate, stopping when no further improvement is made. $Pot_k(X.A)$ is initialized as the set of attributes

```
Let H' be an initial valid hypotheses;
S(H') := score_C(H');
repeat
  H := H';
  S(H) := S(H');
  foreach H" ∈ ρ_g(H') ∪ ρ_s(H') do
     if H" is (logically) valid on C then
        if the Bayesian network induced by H" on the data are acyclic then
           if score_C(H")> S(H') then
              H' := H";
              S(H') := score_C(H");
           end if
        end if
     end if
  end
until S(H')≤ S(H);
return H';
```

Fig. 2. Greedy algorithm for searching the structure of Bayesian logic programs proposed in [13]

in X. In successive phases, $Pot_{k+1}(X.A)$ would consist of all of $Pot_k(X.A)$, as well as all attributes that are related to X via slot chains of length less than k. As the set of potential parents is expanded at each iteration, this can result in a large set. Thus, a refined algorithm that only adds parents to $Pot_{k+1}(X.A)$ if they seem to "add value" beyond $Pot_k(X.A)$ is used. For a detailed explanation see [6].

BLP Learning Algorithm. To learn the qualitative component of a BLP, [13], proposes an ILP approach.

It uses the notion of learning from interpretation to learn the logical structure of a BLP, which should be satisfied by all the examples in the training set.

The algorithm (see figure 2)proceeds proposing modifications on the initial BLP performed by refinements operators. These operators can add (ρ_s) a proposition to the body of a clause or delete (ρ_g) it from the body. Each proposed BLP (hypothesis) has to be consistent with the training set and acyclic. The best structure is chosen using a scoring function.

3 Revision of Bayesian Logic Program

Revision of Bayesian Logic Program (RBLP) is a two-step algorithm. The approximating procedure, proposed in [29], unifies theory revision, based in FORTE to search for a minimal revision of a given BLP consistent with the available

training set, with an adaptation of the algorithm EM [16] for learning the entries of the CPDs. The BLP found has the best probabilistic score for the search space aforementioned. The maximization procedure searches for the highest probabilistic score BLP in the entire search space (restricted to the BLPs consistent with the training set).

In the following sections we show the original RBLP, which is the approximating procedure, and define the maximization procedure.

3.1 RBLP Approximating Procedure

Similarly to BANNER [28] for a propositional Bayesian network, RBLP_AP revises the parameters to improve classification accuracy and if the BLP does not adequately fit the training data then its structure is modified and the parameters are retrained. This process is repeated until additional training does not logically improve the accuracy.

The specific task addressed is:

given: a BLP and a set of training examples (**C**).
find: a revised BLP consistent with **C** and with the highest probabilistic score considering the search space defined by theory revision techniques.

The RBLP_AP top level algorithm (see figure 3) begins with learning the CPDs of the initial BLP and scoring it. The $score_{\mathbf{C}}$ function is responsible for these processes.

[16] and [14] have proposed algorithms for learning the CPDs of a PRM and a BLP, respectively. The former made an adaptation of the EM algorithm and the latter was based in the gradient method [1]. RBLP_AP uses the first approach. First the Bayesian network corresponding to each example must be built. If the query-answering procedure failed in this construction, the example under consideration was not proved from the current BLP. As in FORTE, this example is kept as a misclassified example and the revision points are annotated. This is one kind of misclassification, the other one, the Bayesian network is built for the corresponding example, but this example was proved with low probability, and so misclassified. The only way to correct those misclassification, using RBLP_AP, is with the CPD learning algorithm.

After building all the Bayesian networks and learning the CPDs if unproved examples were found, the structure of the BLP must be revised. This revision is performed based in the same revision operators presented in FORTE but, instead of using counting as a scoring function, RBLP_AP uses a probabilistic scoring function, since it is dealing with a BLP. All the revision points will be used in an attempt to find the best operator to be implemented.

Choosing the Best Revision. The two main scoring function commonly used to learn Bayesian networks are the Bayesian scoring function [8] and the function based on the principle of minimal description length(MDL) [17]. RBLP_AP is

```
Let H' be the initial BLP;
S_BR := score_C(H');
Best_Revision := H';
 if misclassified examples were found
    repeat
      repeat for each revision point
          generate possible revisions (H);
          foreach H" ∈ H do
              if H" improves logically the BLP then
                  S_H" := score_C(H");
                  if S_H" > S_BR then
                      Best_Revision := H";
                      S_BR := S_H";
                  end if
              end if
          end
      until finish the set of revision points
      if Best_Revision improves the BLP
          H' := Best_Revision;
      end if
    until no revision improves logically the BLP
end if
return H';
```

Fig. 3. RBLP_AP's top-level algorithm

currently using the first one ($L(H : \mathbf{C})$), where H is the current qualitative component of the BLP and $\mathbf{C}$ is the set of training examples.

$$L(H : \mathbf{C}) = \mathbf{P}(\mathbf{C}|H, \theta)$$

where θ is the uncertainty parameters (CPDs). Then, the best revision is the one that maximizes the scoring function.

$$Best\_Revision = \max_{H \in \mathbf{H}} L(H : \mathbf{C}) = \max_{H \in \mathbf{H}} \mathbf{P}(\mathbf{C}|H, \theta)$$

where $\mathbf{H}$ is the set of hypotheses, that is, possible BLPs found after applying each possible revision operator in a revision point.

RBLP_AP considers independence between the examples in the training set.

$$P(\mathbf{C}|H, \theta) = \prod_{i \in [1..m]} P(C_i|H, \theta)$$

where $(C_1, ..., C_m)$ are all the examples in the training set.

To maximize the scoring function, first the probability of each example in the subset considered has to be found. The BLP query-answering procedure can be used to find these probabilities, $P(C_i|H, \theta)$, since in its last step it performs inference in the Bayesian network found.

p(Z,X)	p(Z,Y)	**P**(s(X,Y))
T	T	0.99
F	T	0.50
T	F	0.50
F	F	0.50

au(X,Y)	g(X,male)	**P**(u(X,Y))
T	T	0.9514
F	T	0.0006
T	F	0.50
F	F	0.50

Fig. 4. The table on the left shows the CPD for the clause 1 and the table on the right for the clause 3

m(X,Z)	s(Z,W)	p(W,Y)	**P**(au(X,Y))
T	T	T	0.9802
F	T	T	0.50
T	F	T	0.0032
T	T	F	0.50
F	F	T	0.50
T	F	F	0.50
F	T	F	0.50
F	F	F	0.50

Fig. 5. CPD for the clause aunt_uncle(X, Y) | married(X,Z), sibling(Z, W), parent(W,Y)

Example. As an example consider the Bayesian logic program specified below
Qualitative component

1. sibling(X,Y) | parent(Z,X), parent(Z,Y)
2. aunt_uncle(X,Y) | married(X,Z), sibling(Z,W), parent(W,Y).
3. uncle(X,Y) | gender(X, male), aunt_uncle(X,Y).

where the predicates are binary. Using just the initial letters of each predicate.
Quantitative Component

$\mathbf{P}(g(X,\_)) = < 0.95, 0.05 >$ $\mathbf{P}(p(X,Y)) = < 0.85, 0.15 >$
$\mathbf{P}$ $(m(X,Y)) = < 0.90, 0.10 >$

Let's consider that the training set has the three examples given below.

1. u(eric,jane)|g(eric,male),p(ann,kari),p(ann,eric), p(kari,jane)
2. u(bob,jane)|g(bob,male),m(bob,helen),p(ann,helen), p(ann,kari),p(kari,jane)
3. u(john,jane)|g(john,male),m(john,lisa),p(ann,kari), p(ann,lisa), p(kari,jane)

For each evidence in the examples, which its predicate is not a head of some clause in the BLP, a clause for this evidence, with true as the head, is add to the BLP.

Calling the BLP query-answering procedure for the three examples the Bayesian networks shown in figures 6 and 7 are built.

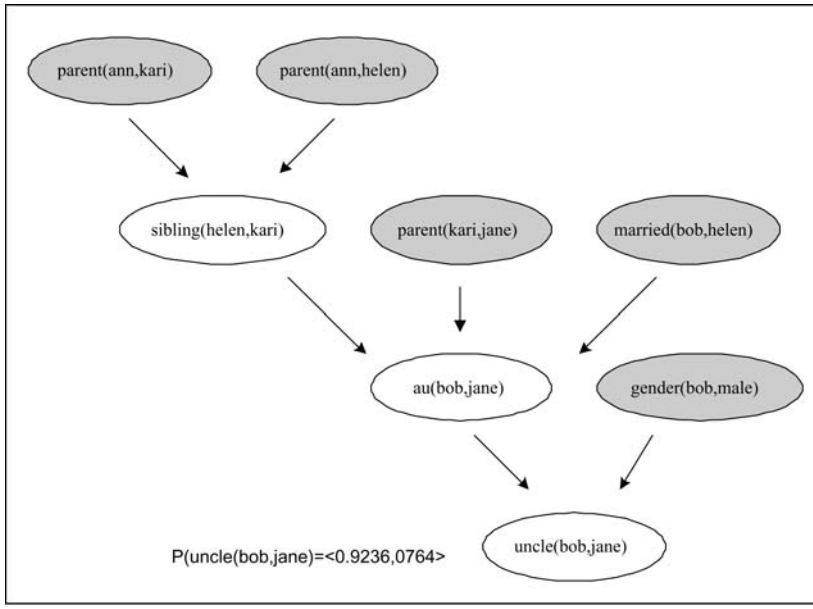


Fig. 6. Bayesian network built from example 2

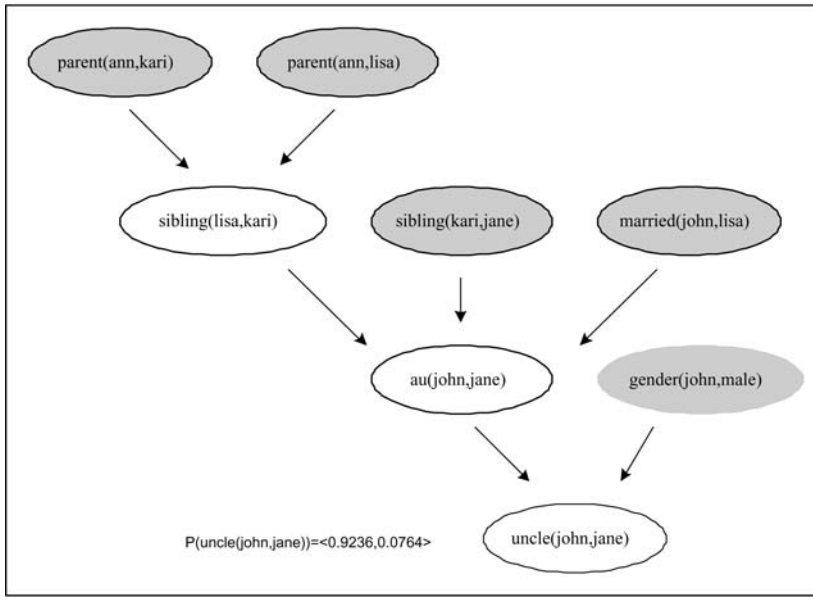


Fig. 7. Bayesian network built from example 3

Notice that the second and the third example were classified correctly, while the first one failed in classification. Since, the failure occurred because the example 1 was not proved, the qualitative component of the BLP needs to be revised.

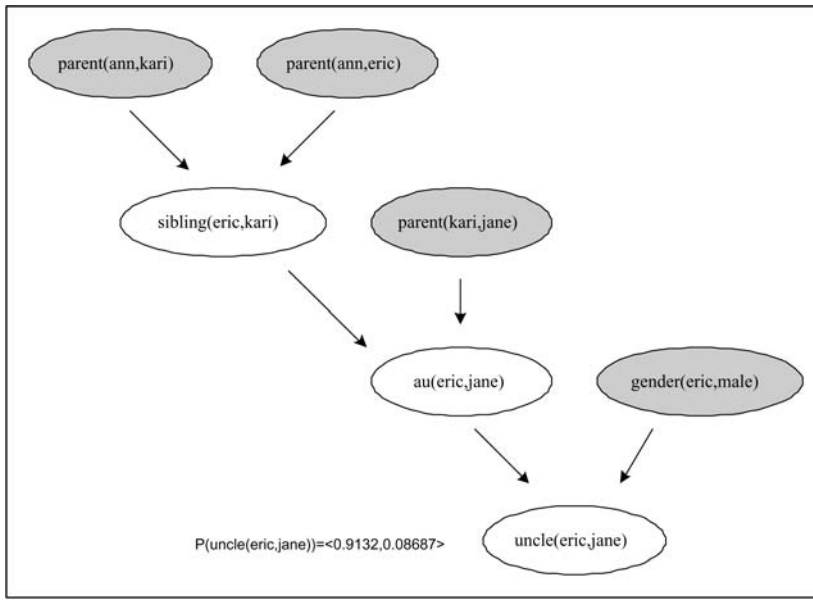


Fig. 8. Bayesian network built from example 1

Table 2. CPD for the clause aunt_uncle(X, Y) | sibling(Z, W), parent(W,Y)

s(Z,W)	p(W,Y)	**P**(au(X,Y))
T	T	0.85
F	T	0.01
T	F	0.01
F	F	0.01

Applying the scoring function on the initial BLP, we find the likelihood below.

$$\mathbf{P}(\mathbf{C}|H_0, \theta) = 0.9019$$

where H_0 is the structure of the initial BLP (clauses 1,2,3).

We need a generalization operator to revise this BLP, which can be: (a) the delete antecedent operator, in this case to delete the antecedent $married(X, Z)$ of the second clause, generating the clause 4;

$$4. \quad aunt\_uncle(X, Y)|sibling(X, Z), parent(Z, Y).$$

(b) or the add new clause operator, which in this example will also be the clause 5.

As a result, the first example is proved and performing again the BLP query-answering procedure, the Bayesian network in figure 8 is built.

p(Z,X)	p(Z,Y)	**P**(s(X,Y))
T	T	0.9967
F	T	0.50
T	F	0.50
F	F	0.50

au(X,Y)	g(X,male)	**P**(u(X,Y))
T	T	0.9669
F	T	0.0066
T	F	0.50
F	F	0.50

Fig. 9. The table on the left shows the CPD for the clause 1 and the table on the right for 3

Table 3. CPD for the clause 4

s(Z,W)	p(W,Y)	**P**(au(X,Y))
T	T	0.9470
F	T	0.0006
T	F	0.50
F	F	0.50

First Revision (Delete antecedent operator)

The CPD shown in table 4 for the clause aunt_uncle(X, Y) | sibling(Z, W), parent(W,Y) is consider as a starting point to the learn parameters procedure for learning the CPD of this clause.

After applying the parameter learning procedure the CPDs shown in figure 9 and table 3 are found.

Applying the scoring function on the current BLP, we find the likelihood below.

$$\mathbf{P}(\mathbf{C}|H_1, \theta) = 0.9783$$

where H_1 is the structure of the BLP found after applying the first operator (clauses 1,3,4).

Since $S_{H_1} > S_{H_0} \therefore Best\_Revision : S_{H_1}$

Second Revision (Add new clause operator)

Considering the same CPD for the clause aunt_uncle(X, Y) | sibling(Z, W), parent(W,Y) as starting point, the same CPDs are found when applying the learning parameters procedure except that this time the CPD for the clause 2 is also learn.

Applying the scoring function:

$$\mathbf{P}(\mathbf{C}|H_2, \theta) = 0.9152$$

where H_2 is the structure of the BLP found after applying the second operator (clauses 1,2,3,4).

Since $S_{H_2} < S_{H_1}$ the best revision will still be S_{H_1} and H_1 will be the hypothesis implemented.

Since no more logical improvement can be performed in the current BLP, the final BLP is:

Table 4. CPD for the clause aunt_uncle(X, Y) | married(X,Z), sibling(Z, W), parent(W,Y)

m(X,Z)	s(Z,W)	p(W,Y)	**P**(au(X,Y))
T	T	T	0.9998
F	T	T	0.50
T	F	T	0.0078
T	T	F	0.50
F	F	T	0.50
T	F	F	0.50
F	T	F	0.50
F	F	F	0.50

1. uncle(X,Y) | gender(X, male), aunt_uncle(X,Y).
3. sibling(X,Y) | parent(Z,X), parent(Z,Y).
4. aunt_uncle(X,Y)| sibling(X,Z), parent(Z,Y).

3.2 RBLP Maximization Procedure

The approximation procedure discussed in the previous section, finds a consistent BLP and with the highest probabilistic score considering the search space composed by structures only modified in the revision points.

For most situations, and specially for a classification task where the initial BLP is approximately correct, the BLP found is sufficient. If this is not the case or the RBLP_AP did not perform the expected improvement in classification and we have the time for a more accurate answer, we propose to extend RBLP by applying the maximization procedure after performing the approximation one. This extension of the original RBLP (RBLP_MP) is based on the BLP learning algorithm.

RBLP_MP attempts to find the structure with the highest probabilistic score in the entire search space, restricted to consistent BLPs and avoiding the RBLP_AP revisions points.

4 Relationship to Other Frameworks

In this section we compare PRM and BLP learning algorithms, and show the relationship between them and RBLP.

We can point out as an important difference between the PRM and BLP learning algorithms, the total probabilistic approach of the former versus the logical-probabilistic approach of the latter. The algorithm to learn a PRM does not guarantee that the structure found is consistent with the training set; it is concerned about finding the highest probabilistic scoring structure. On the other hand, the algorithm to learn the structure of a BLP necessarily finds one consistent with the training data which is also maximum probabilistically.

An important difference between RBLP_AP and PRM learning algorithms is the same as the aforementioned between BLP and PRM learning algorithms, since it also guarantees that the final BLP is consistent with the training set.

As we have seen in section 2.3, BLP and PRM learning algorithms starts with a initial structure and propose modifications on its entire structure, choosing the one with the highest score. For a classification task, if the initial theory is approximately correct, it would be more efficient to propose modifications only in those places responsible for the misclassifications, analyzing these possible revisions and choosing the one with the highest probabilistic score. Therefore, theory revision would be a good choice, since it identifies the places to be repaired using the training examples and modifies only those. The theory found has the best probabilistic score for the search space aforementioned. This is another important difference between these two approaches and RBLP_AP. Under the consideration of an approximately correct theory, the BLP found by the RBLP_AP and the BLP learning algorithms would not differ much, but if one is interested, or the RBLP_AP did not perform the expected improvement in classification, and have time, the RBLP_MP can be used to search for the highest probabilistic score BLP in the entire search space (restricted to the theories consistent with the training set). If the initial BLP is not approximately correct, it is more efficient to directly perform the RBLP_MP. The results will be similar to the the ones given by the BLP learning algorithm but without any efficiency improvement.

5 Conclusion

In this paper, we presented an integration of techniques to learn the CPDs and to modify the structure of a first-order Bayesian Network. The resulting Revision of Bayesian Logic Programs (RBLP)algorithm merges Bayesian Networks with Inductive Logic Programming (Theory Revision). The use of BLP to represent First-order Bayesian Networks allowed this integration to be smoother.

RBLP is a two-step algorithm. The approximating procedure finds a consistent BLP and with the highest probabilistic score considering the search space composed by structures only modified in the points responsible for the misclassifications. The maximization procedure, which is basically [13] BLP learning algorithm, searches for the highest probabilistic score BLP in the entire search space (restricted to the BLPs consistent with the training data).

The PRMs and BLPs structure learning algorithms were compared. While the former is interested in the structure with the highest probability not worrying about consistency, the latter guarantees to find a structure consistent with the training set.

The most important contribution was that RBLP_AP, when dealing with an initial first-order approximately correct Bayesian classifier, can perform more efficiently than the two algorithms aforementioned. For other cases, RBLP_MP was proposed obtaining similar results to the BLP learning algorithm.

The idea of an approximating procedure can be similarly applied to PRMs. In the future we will perform more extensive experiments including the use of a MDL scoring function.

Acknowledgements

The authors are partially financially supported by the Brazilian Research Council, CNPq.

References

1. J. Binder, D. Koller, S. Russell and others. Adaptive probabilistic networks with hidden variables. Machine Learining. pp.213-244, 1997.
2. C.A. Brunk, 1996. An Investigation of knowledge Intensive Approaches to Concept Leaning and Theory Refinement. Ph.D. thesis, University of California, Irvine, CA.
3. D.M. Chickering. Learnig Bayesian networks is NP-complete. In: Learning from Data: Artificial Intelligence and Statiscs V. D. Fisher and H.-J. Lenz. Spring Verlag, pp.121-130, 1996.
4. N. Friedman, I. Nachman and D. Peér. Learning Bayesian Network Structure from Massive Datasets: The Sparse Candidate Algorithm. In Proceedings of the Fifteenth Conference on Uncertainty in Articial Intelligence. pp.206-215, 1999.
5. I. Fabian and D. A. Lambert. First-Order Bayesian Reasoning. Proceedings Eleventh Australian Joint Conference on Artificial Intelligence. 1502, 131-142, 1998.
6. N. Friedman, L. Getoor , D. Koller and A. Pfeffer. Learning Probabilistic Relational Models. Proc. IJCAI-99, 1300-1309, 1999.
7. P. Haddawy. An overview of some recent developments on Bayesian problem soving techniques. AI Magazine, Spring 1999 - Special issue on Uncertainty in AI, 1999.
8. D. Heckerman. A tutorial on learning with Bayesian networks. In M.I. Jordan, editor, Learning in Graphical Models, pp 301- 354. MIT Press, Cambridge, MA, 1998.
9. M. Jaeger. Relational Bayesian Networks. Proc. 13th Conference on Uncertainty in AI. pp.266–273. Morgan Kaufmann, 1997.
10. K. Kersting, L. De Raedt and S. Kramer. Interpreting Bayesian Logic Programs. In Working Notes of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data (SRL), Austin, Texas, 2000.
11. K. Kersting and L. De Raedt. Bayesian Logic Programs. Proc. of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming, 2000.
12. K. Kersting and L. De Raedt. Bayesian Logic Programs. Technical Report 151, University of Freiburg, Institute for COmputer Science, Abril 2001.
13. K. Kersting and L. De Raedt. Towards Combining Inductive Logic Programming with Bayesian Networks. In proceedings of the Eleventh International Conference of Inductive Logic Programming. Strasbourg, France, pp. 118-131, 2001.
14. K. Kersting and L. De Raedt. Adaptive Bayesian Logic Programs. In proceedings of the Eleventh International Conference of Inductive Logic Programming. Strasbourg, France, pp. 104-117, 2001.
15. D. Koller. Probabilistic Relational Models. In Proc. of the 9th Int. Workshop on ILP. LNAI 1634, 3-13, Springer Verlag, 1999.
16. D. Koller and A. Pfeffer. Learning probabilities for noisy first-order rules. In Proc. of the 15th International Joint Conference on Artficial Intelligence (IJCAI-97), 1316-1323, 1997.

17. W.Lam and F.Bacchus. Learning Bayesian belief networks: An approach based on the mdl principle. Computatinal Intelligence, 10(3),pp. 269-29, 1994.
18. S.L Lauritzen. The Em algorithm for graphical association models with missing data. Computational Statistics and Data Analysis, 19: 191-201, 1995.
19. J. Lloyd. Foundations of Logic Programming. Springer Verlag, 2. edition, 1989.
20. S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, Advances in Inductive Logic Programming. IOS Press, 1996.
21. G.J. McLachlan and T. Krishnan. The EM algorithm and Extensions. Wiley Interscience, 1997.
22. T. Mitchell. Machine Learning. McGraw-Hill, New York, NY, 1997.
23. L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. Theoretical Computer Science, 171:147-177, 1997.
24. J. Pearl. Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufman, 2. edition, 1991.
25. D. Poole. Probabilistic horn abduction and bayesian networks. Artificial Intelligence, 64(1): 81-129, 1993.
26. D. Poole. Abducing Through Negation as Failure: Stable models within the independent choice logic. Journal of Logic Programming, 44: 5-359, 2000.
27. D. Poole. Learning, Bayesian Probability, Graphical Models, and Abduction. P. Flach and A. Kakas, editors, Abduction and Induction: essays on their relation and integration. Kluwer, 1998.
28. S. Ramachandran and R. Mooney. Theory Refinement of Bayesian Networds with Hidden Variables. 15th International Conference on Machine Learning (ICML), pp.454-462, Morgan Kaufman, 1998.
29. K. Revoredo and G. Zaverucha. Theory Refinement of Bayesian Logic Programs. Proceedings of the Eigth International Conference on Neural Information Processing (ICONIP), Shanghai, China. pp. 1088-1092, 2001.
30. B. Richards and R. Mooney. Automated Refinement of First-Order Horn-Clause Domain Theories. Machine Learning 19, pp. 95-131, 1995.
31. S. Wrobel. First-order Theory Refinement. Advances in Inductive Logic Programming, edited by Luc de Raedt, pp. 14-33, IOS Press, 1996.

The Applicability to ILP of Results Concerning the Ordering of Binomial Populations

Ashwin Srinivasan

Oxford University Computing Laboratory, Wolfson Bldg., Parks Rd, Oxford, UK

Abstract. Results obtained in the 1950's on the ordering of k binomial populations are directly applicable to computing the size of data sample required to identify the best amongst k theories in an ILP search. The basic problem considered is that of selecting the best one of k ($k \geq 2$) binomial populations on the basis of the observed frequency of 'success' on n observations. In the statistical formulation of the problem the experimenter specifies that he or she would like to guarantee that the probability of correct selection is at least P^* ($0 \leq P^* < 1$) whenever the true difference between the best and second best population is at least d^* ($0 < d^* \leq 1$). The principal result of interest is that for large values of k, the value of n required to meet any fixed specification (d^*, P^*) is approximately equal to some constant multiple of ln k. Depending on the k, d^* and P^* specified, this value of n can be substantially lower than that obtained within the corresponding PAC formulation. Applied to the search conducted by an ILP algorithm, the result guarantees with confidence level P^*, that the true accuracy of the theory selected will be no more than d^* below that of the best theory. In this paper, we review this basic result and test its predictions empirically. We also comment on the relevance to ILP of more recent work in the field.

1 Introduction

Viewing description learning as entailing a search through a hypothesis space naturally raises issues related to ranking and selection. For many contemporary ILP systems these ultimately concern the ranking and selection of theories, with the aim being to identify the best theory according to some performance measure. How much computational effort should we expend in identifying this theory? Do we need to have very precise estimates of the performance of each theory? How confident are we that the theory we have selected is really the best one? These are questions of no little interest to ILP developers and practitioners alike.

There is a substantial body of statistical literature [3,6] devoted to the study of the following (and related problems):

Best of k. *Suppose there are k populations. Population i has parameter value θ_i. A sample of n observations from each population is used to yield an unbiased estimate $\hat{\theta}_i$ for θ_i. We wish to ensure, with high probability, that the population with the best estimate is also the population with best parameter value.*

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 238–253, 2003.

The principal statistical concern is to determine the value of n for 'correct selection'. For Binomial populations, the assumptions made are the following:

1. Observations (successes or failures) from the same or different populations are independent.
2. For a given population Π_i, the probability of success p_i is fixed.
3. It can be determined without error whether a success or failure has occurred.

The relevance to ILP is evident if we view each of the k populations as the totality of observations about the performance of one of k theories in a hypothesis space; and that the primary task of the ILP system is to select the best amongst these theories[1]. When restricted to the problem of classification, it is often reasonable to assert that the best theory is the one with the highest probability of correctly classifying an example (that is, the best theory is the most accurate one). In this case, we can take each theory as representing a binomial population parametrised by a probability of success equal to the true accuracy of the theory. A solution to the 'best of k' problem posed earlier will then automatically yield a solution to the problem of selecting the theory with the highest accuracy, provided the corresponding assumptions hold. Of these, Assumptions (2) and (3) would appear to be reasonable. Assumption (1) requires closer inspection, since the theories in a search are typically *not* independent. What is required by the assumption is that the outcome of one trial (here, the result of a theory classifying a randomly drawn example) does not affect the outcome of another (that is, the result of a theory classifying another randomly drawn example).

An immediate question that arises is this: is there a solution to the 'best of k' problem within the PAC [16] framework? The answer is: yes, not within the standard PAC model, but within a generalisation termed 'agnostic PAC learning' [10, p.210] However, the approach here can result in lower values of n without compromising any 'worst case' assumptions. More recently, a sequential ranking and selection procedure has been proposed by Scheffer and Wrobel [12]. For the special case of selecting amongst Binomial populations based on success probabilities, their upper bound on the number of observations required is the same as that reached within the agnostic PAC model.

At this juncture, it is instructive to point out that the problem considered here is essentially one of *ranking* as opposed to one of *estimation*. The latter is concerned with the sampling requirements for estimating, to some degree of precision, the value of a parameter. In order to determine the best of a set of alternatives, it is often not necessary to obtain very precise estimates for the

[1] Many ILP systems construct theories using a greedy procedure by searching for locally best single clause additions. There is nothing in the mathematics presented here that restricts us to complete theories. In Section 2.1, we examine how the results can be used to select the locally best single-clause addition. Whether or not a greedy search is used, the first-order setting adopted in ILP can result in a hypothesis space with infinitely many elements. Here, we are concerned more with practical implementations that are restricted to exploring some finite subset of that space.

parameter of interest[2] (of course, once the best has been determined, significant effort can *then* be invested in estimating its parameter value). Within ILP, this is utilised indirectly by the estimation procedures of [13] and [15]. In both cases, estimates obtained are imprecise (for different reasons), but the overall experimental results appear to be good, probably for the reason just mentioned. If selection amongst alternatives is the prime goal, then it is more efficient to use statistical approaches that directly address the problem of ranking and selection.

The results utilised in this paper use the so-called indifference zone approach to address the 'best of k' problem. The intuitions underlying the approach are these: (a) inherent randomness in the observed values implies that we can never be *absolutely* sure of making the correct choice. We will therefore have to qualify our choice with a probability value; and (b) errors of selection are more likely amongst populations close to the best (for example, we are very unlikely to erroneously select the worst population in preference to the best). In particular if the (true) difference between the best and second best populations is small enough, then the error involved in selecting the second best is one of little consequence. Equally, there is little to be gained in investing a large effort—usually related to the sample size—directed at resolving such small differences.

This paper is organised as follows. In Section 2 we describe the indifference zone method applied to the specific problem of selecting the best of k binomial populations. Section 3 tests empirically the predictions made by the principal result in Section 2. Section 4 concludes the paper with pointers to further work on the topic of ranking and selection that is of particular interest to ILP. Appendix A presents an independently derived result of related interest.

2 Selecting the Best Binomial Population

The Binomial distribution is an appropriate model for problems where the probability of observing an event E (usually called 'success') is some value p for each of n independent trials. Then, the probability that E will be observed in exactly x of these trials is given by the Binomial distribution. The problem of selecting the best of k binomial populations is then the following (from [14]). Each of k binomial populations Π_i is associated with a fixed probability of success p_i where $0 \leq p_i \leq 1$ $(i = 1, 2, \ldots, k)$. Let the ordered values of the p_i be denoted by

$$p_{[1]} \geq p_{[2]} \geq \cdots \geq p_{[k]}$$

That is, the population associated with $p_{[1]}$ is the 'best population.' No *a priori* information is assumed about the values of the p_i or about the pairing of the Π_i with the $p_{[j]}$ $(1 \leq i, j \leq k)$. Specifically, we do not know which of the populations Π_i correspond to $p_{[1]}$. The goal is to select the population associated with $p_{[1]}$ [3] on the basis of the observed frequency of successes on n observations drawn from each population. This event will be called 'correct selection' or CS.

[2] Elsewhere [9] this has been noted as "order converging faster than value."

[3] If there are t ties for the first place, then selection of any one of the associated t populations will suffice.

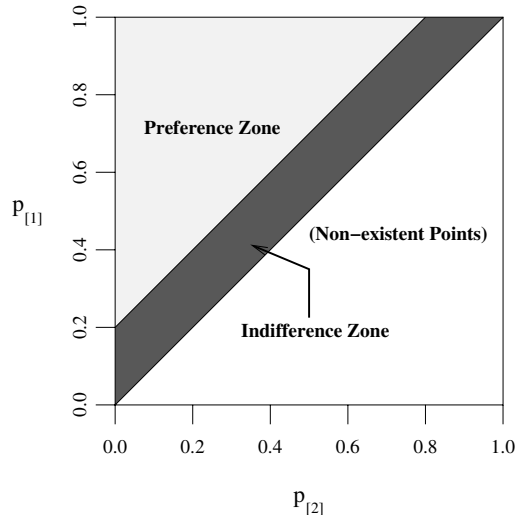


Fig. 1. Indifference and preference zones in parameter space for $d^* = 0.2$. We wish to ensure, with high probability, that the best population is selected in the region marked by the preference zone.

The 'indifference zone' approach (first described for normal populations by [2]) proceeds on the basis that if the the best and the second best populations (associated with $p_{[1]}$ and $p_{[2]}$ respectively) are 'close enough' then the error in wrongly selecting the second best population is of little consequence. That is, we are indifferent as to which amongst the top two populations are selected. In [14], the difference $d = p_{[1]} - p_{[2]}$ is used as the measure of the distance between the two populations. Then, the indifference zone consists of those values of $p_{[1]}$ and $p_{[2]}$ for which d is less than some (pre-specified) value d^* (d^* can be thought of the smallest difference 'worth detecting' [3]). Conversely, the values of $d \geq d^*$ can be thought of as belonging to a *preference zone* where we prefer to select the correct population. These are shown diagrammatically in Fig. 1 for $d^* = 0.2$. This requirement can be encapsulated in probabilistic terms as the following:

Suppose we are given the constants d^* $(0 < d^* \leq 1)$ and P^* $(0 \leq P^* < 1)$. Let $P_{CS} = P_{CS}(p_{[1]}, \ldots, p_{[k]})$ denote the probability of a correct selection based on the frequency of success on n observations and $d = p_{[1]} - p_{[2]}$. Then we require

$$P_{CS} \geq P^* \quad \text{for} \quad d \geq d^* \tag{1}$$

(We may wish to stipulate that $P^* > 1/k$. Otherwise, a trivial selection procedure that randomly selects amongst the k populations would suffice).

The final selection will be made on the observed frequency of success and the only remaining problem is to determine the value of the number of observations n for which (1) is satisfied for a given pair of values (d^*, P^*). The key notion used is that of the 'least favourable configuration.'

Least Favourable Configuration

Recall that $P_{CS} = P_{CS}(p_{[1]}, \ldots, p_{[k]})$ denotes the probability that the (true) best population also has the highest frequency of success on n observations. It is intuitively clear that for given values of d^*, n, k some configurations $(p_{[1]}, \ldots, p_{[k]})$ would result in higher values for P_{CS} than others. That is, these are 'easier' configurations, where it is less likely to make the wrong choice. Of all configurations, the one that results in the lowest P_{CS} is called the *least favourable configuration* or LFC. Let this value of P_{CS} be denoted by P^L_{CS}. The task is to find the value of n such that $P^L_{CS} \geq P^*$, for it would follow from this that $P_{CS} \geq P^*$ for the same value of n. In [14], it is shown that that the probability of correct selection is smallest for the configuration:

$$p_{[1]} - d^* = p_{[2]} = p_{[3]} = \cdots = p_{[k]} \tag{2}$$

This does not, of course, locate the actual value of $p_{[1]}$ for which P_{CS} is smallest. P_{CS} may be regarded as a function of $p_{[1]} = p_{[1]}(d; n)$, which assumes a minimum value P^L_{CS} for $d = d^*$. If the value of $p_{[1]}$ where this occurs is denoted by $p^L_{[1]}$, then P^L_{CS} is given by [14] (we do not reproduce any of their proofs here, as it is their results that are of principal importance):

$$P^L_{CS} = \sum_{j=0}^{n} b_{1j} \sum_{i=0}^{k-1} \frac{C_i^{k-1}}{1+i} b_{2j}^i B_{2,j-1}^{k-1-i} \tag{3}$$

where $B_{2,-1} = 0$ and:

$$\begin{aligned} b_{1j} &= C_j^n (p^L_{[1]})^j (q^L_{[1]})^{n-j} && (0 \leq j \leq n) \\ b_{2j} &= C_j^n (p^L_{[1]} - d^*)^j (q^L_{[1]} + d^*)^{n-j} && (0 \leq j \leq n) \\ B_{2,j} &= \sum_{i=0}^{j} b_{2j} \end{aligned}$$

Here $q^L_{[1]} = 1 - p^L_{[1]}$. It is further shown that for $n \geq 10$

$$p^L_{[1]} = (1 + d^*)/2 \tag{4}$$

can be used in computing from (3). Thus, given values for d^*, P^*, we can use (3) and (4) to obtain a value of n such that $P^L_{CS} = P^*$. This value of n would ensure that $P_{CS} \geq P^*$ for any configuration. The computations in (3) are intensive and in [14] it is shown that for $n \geq 10$, the resulting value is closely approximated by:

$$n \approx \frac{B}{{d^*}^2} \tag{5}$$

where B is a constant depending on P^* and k. In fact, it is shown that for large k, this approximation is a straight line proportional to $\ln k$. The approximation is

k	$d^* = 0.2, P^* = 0.95$	$d^* = 0.1, P^* = 0.95$	$d^* = 0.1, P^* = 0.90$	$d^* = 0.1, P^* = 0.99$
10^1	74	293	223	451
10^2	123	492	404	678
10^3	173	692	586	904
10^4	223	892	767	1131
10^5	273	1092	949	1358
10^6	323	1291	1130	1584

Fig. 2. Approximate values of n required to ensure with probability P^* that the best amongst k theories is selected, with the qualification that theories with (true) accuracy no more than d^* below that of the best theory would also be acceptable.

also empirically found to over-estimate slightly the true value of n. Fig. 2 shows some illustrative values of n obtained from the approximation for different values of k, d^*, P^* (a Prolog program that implements (3) and (5) is available from the author). For large k, n is more sensitive to changes in d^* than to changes in P^*. This is illustrated by the slopes of the lines in Fig. 3. These results are used as the basis for the following single-stage sampling procedure (from [3]):

Procedure $\mathcal{B}_{SH}$. Given k, d^*, P^* identify the binomial population associated with $p_{[1]}$.

1. Take a random sample of n (obtained from (3) s.t. $P^L_{CS} = P^*$) observations from each of the k populations, and record the number of successes (that is, the number of correct classifications) observed for each population.
2. Select the population that yielded the highest number of successes as the one associated with $p_{[1]}$. In case of ties, select randomly amongst those with the maximum number of successes.

In [8] it is proved that $\mathcal{B}_{SH}$ is optimal in the sense that no other single-stage sampling procedure with fewer observations could guarantee the probability requirements of (1).

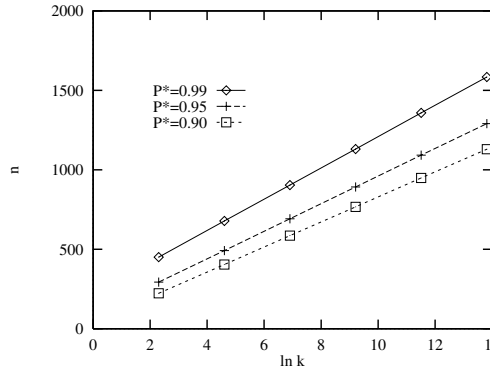
A related aspect to the calculation of sample size is the calculation of the *operating characteristic curve*. This assumes that the sample size n is fixed (this may be the case in practice), and examines the relation of P^*, computed using (3), to d^*. This would indicate directly the trade-off between confidence and precision that would have to be made for a particular value of n. Figure 4 illustrates this.

Confidence Statement

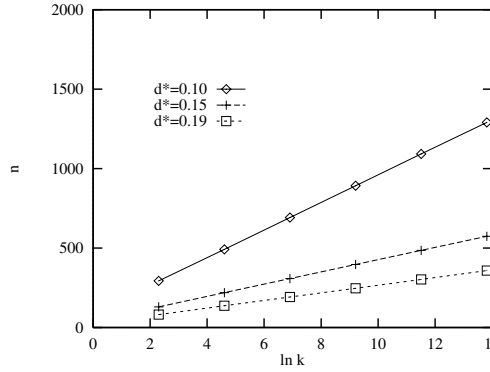
Let p_s be the p-value of the population selected using Procedure $\mathcal{B}_{SH}$. Then, the results above can be re-formulated as follows:

$$p_{[1]} - d^* \leq p_s \leq p_{[1]} \quad \text{with confidence at least } P^* \tag{6}$$

That is, with confidence level P^*, we can be sure that the p-value of the selected population is no worse than d^* below that of the best population.



(a) $d^* = 0.10$



(b) $P^* = 0.95$

Fig. 3. Changes in values of n.

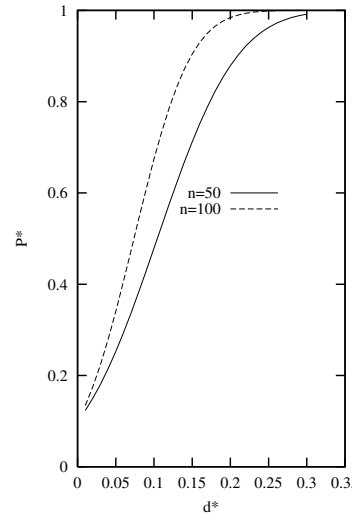


Fig. 4. Operating characteristic curves for $k = 10$.

2.1 Application to ILP

We will consider a straightforward application of these results to the problem of selecting the best of k theories in a hypothesis space, using a sample of n examples. It is assumed that the selection procedure will proceed as in $\mathcal{B}_{SH}$, with one small modification: namely, that we shall content ourselves with using (5) to calculate the value of n. Since this value has been observed to yield an over-estimate on the true value of n, the modified procedure $\mathcal{B}'_{SH}$ will be sub-optimal, but still correct. For completeness, this procedure is as follows:

Procedure $\mathcal{B}'_{SH}$. Given k theories, d^*, P^* identify the binomial population associated with $p_{[1]}$.

1. Take a random sample of n (minimum of: the value from (5) s.t. $P^L_{CS} = P^*$ and 10) observations from each of the k populations, and record the number of successes (that is, the number of correct classifications) observed for each population.
2. Select the theory that yielded the highest number of correct classifications as the one associated with $p_{[1]}$. In case of ties, select randomly amongst those with the maximum number of correct classifications.

Selection in a Greedy Search. Many ILP systems construct a theory by employing a greedy search that repeatedly selects the clause that is locally 'best'. When attempting to identify the theory with the best accuracy, this translates to selecting, at each stage, the clause that yields the greatest increase in accuracy. We assume an iterative search algorithm that, on each iteration, examines (at most) k clauses to add to an existing theory H. The selection task is to find the best amongst the k theories that result by adding, in turn, each of the k clauses to H. This is readily addressed by Procedure $\mathcal{B}'_{SH}$, which will guarantee (probabilistically speaking) the best local choice. Of course, as with all greedy searches, there is no guarantee that this would lead to best theory overall.

2.2 Comparison with the Agnostic PAC Bound

'Agnostic' PAC generalises the standard PAC framework by not requiring the target theory to be contained in a learner's hypothesis space. Then the (true) error of theories examined by the learner may be non-zero, and the learner returns the theory with the lowest error on n observations (as in Procedure $\mathcal{B}'_{SH}$). The primary interest in the agnostic setting is to determine the value of n such that the observed error of this selected theory is within some (small) value of its true error. We can use the resulting bound to compute further, with confidence P^*, that the true error of the selected theory is within d^* of the best theory in the hypothesis space. The bound on n obtained is identical to that obtained in Appendix A:

$$n \geq \frac{2 \ln\left(\frac{2k}{1-P^*}\right)}{{d^*}^2} \tag{7}$$

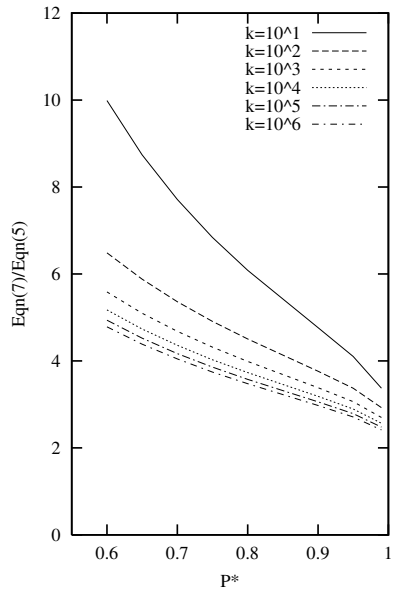


Fig. 5. Ratio of values of n using the agnostic PAC model (Equation (7)) and indifference zone approach (Equation (5)).

Here, k is now the size of the hypothesis space and d^*, P^* correspond to the familiar PAC quantities ϵ and δ respectively. This same value is computed to be an upper bound on the number of observations needed by the sequential sampling procedure in [12]. It is instructive to compare the values obtained from (5) and (7). As there is no simple analytical formula for the constant B in (5), a graphical comparison is presented in Fig. 5. The comparison is independent of d^* since the common factor of $1/{d^*}^2$ is eliminated in the ratio. The graph suggests that while values of n from (7) can be up to 10 times higher than those from (5), for realistic values of k, d^* and P^* the ratio is probably closer to 2 or 3.

3 Empirical Evaluation

Our principal goal is to evaluate the extent to which $\mathcal{B}'_{SH}$ complies with the confidence-statement formulation in (6), namely:

$$p_{[1]} - d^* \leq p_s \leq p_{[1]} \quad \text{with confidence at least } P^*$$

The evaluation is conducted in the form of an empirical study using a large ILP dataset. For this, we will observe the proportion of trials on which the relation above holds for different values of k, d^* and P^*.

3.1 Materials and Method

For ILP, the main use of the results here is in problems with a large amount of example data. For these, we are able to formulate a confidence statement on the selection of the best theory (or locally best theory, if used in a greedy search) by utilising samples of the data. We intend to test empirically the validity of the confidence statement. To do this we will use the standard problem of classifying illegal positions in the KRK chess endgame, using as background knowledge ($\mathcal{B}$) predicates describing the geometry of the chess-board (rank and file adjacency, and rank and file comparison using the standard $<$ relation). This domain has become something of a benchmark test for ILP systems (see [1] for a review). It has the advantage of providing ready access to a large example set (the total number of examples is $8^6 = 262,144$) and being sufficiently amenable to controlled experiments. We focus on using the results in a greedy search and adopt the following method:

Given an initial theory H, for each value of (k, d^*, P^*):

1. Obtain k clauses from the hypothesis space and construct k theories $H_1, \ldots, H_k$ by adding each clause to H.
2. Let $p_1, \ldots, p_k$ denote the true accuracies of $H_1, \ldots, H_k$ and $p_{[1]}$ denote the highest true accuracy.
3. Repeat T times:
   (a) Let p_s be the true accuracy of the theory selected by ${\mathcal{B}'}_{SH}$ when given $H_1, \ldots, H_k$, d^*, and P^*.
   (b) If $p_{[1]} - d^* \leq p_s \leq p_{[1]}$ then record 'correct selection', else record 'incorrect selection'.
4. Let S be the number of correct selections recorded in Step 3b. Compare S/T to P^*.

The following details are relevant:

(a) For simplicity, we will take H to be the background knowledge $\mathcal{B}$. Further, we only consider the following values of $k : 10, 100, 1000$; $d^* : 0.05, 0.10$; and $P^* : 0.95, 0.99$.

(b) In order to test the validity of the confidence statement, we need access to the true accuracies $p_{[1]}$ and p_s of the best and selected theories, respectively. For the KRK domain, it is actually possible to obtain these by evaluating them on the entire database of $262,144$ examples.

(c) 4 different mechanisms for obtaining clauses have suggested themselves to us for Step 1: (i) a simple random sample of k clauses from the hypothesis space; (ii) k clauses in sequence from a systematic ILP search; (iii) k clauses that result in an 'unfavourable configuration'. This results when theories other than the best one lie as close as possible to the edge dividing the indifference and preference zones (see Fig. 1); and (iv) k clauses that result in the least favourable configuration (LFC). For $n \geq 10$, the LFC results when $p_{[1]} = (1 + d^*)/2$ and $p_{[2]} = p_{[3]} = \cdots = p_{[k]} = (1 - d^*)/2$ [3]. Of these, we believe (ii) and (iii) are of definite interest to ILP practitioners.

k	P^*	
	0.95	0.99
10	1.00	1.00
100	1.00	1.00
1000	1.00	1.00

(a) $d^* = 0.05$

k	P^*	
	0.95	0.99
10	1.00	1.00
100	1.00	1.00
1000	1.00	1.00

(b) $d^* = 0.10$

Theories from a systematic search

k	P^*	
	0.95	0.99
10	0.97	1.00
100	1.00	1.00
1000	1.00	1.00

(a) $d^* = 0.05$

k	P^*	
	0.95	0.99
10	0.97	1.00
100	0.97	1.00
1000	1.00	1.00

(b) $d^* = 0.10$

Theories in an 'unfavourable' configuration

Fig. 6. Relative frequency with which true accuracy of the selected theory was within d^* of the true accuracy of the best theory. These are proportions on 30 trials. Thus, an entry 1.00 results when a 'correct selection' was made on all 30 trials. An entry 0.97 indicates that a correct selection was made on 29 trials.

(i) is unrepresentative of how most ILP systems obtain clauses and (iv) is unlikely to occur in practice (especially when k is large). Consequently we will consider only categories (ii) and (iii) in experiments. In particular, for (ii) we will obtain clauses from a breadth-first general-to-specific search using appropriate parameter settings for the Aleph system. For (iii) we first obtain one clause at random. The true accuracy of the corresponding theory is taken to be $p_{[1]}$. The remaining $k-1$ clauses are then obtained by ensuring that the corresponding theories have accuracies as close as possible (but not equal) to $p_{[1]} - d^*$.

(d) For our experiments, $T = 30$.

3.2 Experimental Results

Figure 6 tabulates the relative frequency with which each of the selected theory was found to be within d^* of the best theory. Figure 7 tabulates the corresponding values of n (sample size) used. It is evident that in all cases, the observed relative frequency of correct selection exceeds the value of P^*, which is consistent with the predictions made by the confidence statement (6). On occasions, an incorrect selection is made when theories are in an unfavourable configuration. This is not

k	P^*	
	0.95	0.99
10	1169	1803
100	1968	2710
1000	2767	3616

(a) $d^* = 0.05$

k	P^*	
	0.95	0.99
10	293	451
100	492	678
1000	692	904

(b) $d^* = 0.10$

Fig. 7. Sample sizes used for experiments. Thus, an entry of 1169 for $k = 10, P^* = 0.95$ indicates that the accuracy of each of the 10 theories was estimated using a randomly drawn sample of 1169 examples.

k	Systematic	Unfavourable
10	1	1
100	41	1
1000	9	1

(a) $d^* = 0.05$

k	Systematic	Unfavourable
10	1	1
100	41	1
1000	61	1

(b) $d^* = 0.10$

Fig. 8. Number of theories in the indifference zone. The larger this number, the easier it is to make a correct selection.

surprising, as in these cases, all except one theory (the best one) lie just outside the indifference zone. The sample-guided selection procedure thus has no margin for error. In contrast, theories generated from the systematic search were often in more benign configurations (see Fig. 8).

4 Concluding Remarks

The primary intent of this paper was to demonstrate that results from statistical studies of ranking and selection have a natural applicability to the search problem confronting ILP systems. In the paper, we have utilised the earliest, (and simplest) statistical formulation that yields a single stage sampling procedure that naturally incorporates issues of confidence in the selection and indifference to small deviations from optimality. The sample size is empirically found to be lower than that from the appropriate PAC model, without requiring any prior knowledge about the accuracies of theories being compared. In addition, we believe that ranking and selection techniques offers some interesting conceptual insights, like that of least favourable configurations, screening and sequential sampling (see below), that are not immediately apparent within the PAC model.

We have focussed on one analytical aspect of the problem of selecting the best population, namely, determination of sample size. Elsewhere [4] it has been persuasively argued that for some problems, theorem-proving can be extremely difficult. While sampling in the manner here does not address that problem, it

may mitigate its effects as theorem proving is restricted to a small subset of the data.

There are a number of ways in which the work here can be extended. Some of these are:

1. *Other performance measures*. We have concentrated on selection based on accuracy. ILP practitioners may be interested in other measures of performance (for example, see [11]). Ranking and selection procedures have been developed for distributions other than the Binomial [3,6]. It is clearly of interest to determine the extent to which these results can be used in ILP.
2. *Selection based on screening*. The indifference zone approach is only one possible formulation of the selection problem. An alternative, sometimes called 'subset selection', aims to select a (small) subset of the populations that is guaranteed with high probability to contain the best one [7]. The result is a kind of screening of populations to identify and discard 'inferior' ones. This can be of special interest to ILP when trying to identify a set of possible hypotheses.
3. *Informed sampling*. The sampling requirements derived in the paper are only optimal for single-stage sampling. However, these are obtained to be adequate for the least favourable configuration, which may never occur in practice (see, for example, the configurations that occur during the systematic search in Fig. 8). As a result, the values of n, although lower than those from the PAC model, are still extremely conservative. We can usually do much better if we have some prior information about the configurations that actually occur, or are allowed some preliminary experimentation. The latter approach forms the basis of sequential sampling procedures which have been developed extensively in the field of discrete-event simulation [5]. In ILP, this will allow us to focus computational resources on theories that are more likely to be close to the best.
4. *Noise*. All the results presented here have assumed noise free data (recall Assumption 3 on p.239). It is interesting to see how the approach extends when corrupted by random classification noise.

On the empirical side there is of course value in demonstrating, both that the theoretical predictions are borne out on other ILP datasets, and that good selections result when used by ILP systems. Of special interest on the latter front is the use of these methods in selecting accurate clauses on large-scale industrial and scientific databases. The numbers of examples available in such problems is so large that some principled form of sub-sampling would appear to be extremely desirable. The indifference-zone approach described here provides one possible framework for techniques that are directly relevant to dealing with such issues in ILP.

Acknowledgements

The study of ranking and selection methods was conducted by the author when visiting the School of Computer Science and Engineering, University of New

South Wales, Australia. Special thanks are due to the members of that school, especially Michael Bain, Ashesh Mahidadia, and Claude Sammut. The author is extemely grateful to David Page and James Cussens for acting as patient mathematical oracles, to Khalid Khan for pointing out the relevance of the agnostic PAC setting, and to Filip Zelezny and Steve Moyle who were generous with ideas and assistance.

References

1. M. Bain and A. Srinivasan. Inductive logic programming with large-scale unstructured data. In D. Michie S. Muggleton and K. Furukawa, editors, *Machine Intelligence 14*, pages 233–267. Oxford University Press, Oxford, 1995.
2. R.E. Bechhofer. A single-sample multiple-decision procedure for ranking means of normal populations with known variances. *Annals of Mathematical Statistics*, 25:16–39, 1954.
3. R.E. Bechhofer, T.J. Santner, and D.M. Goldsman. *Design and Analysis of Experiments for Statistical Selection, Screening, and Multiple Comparisons*. John Wiley & Sons, New York, 1995.
4. M. Botta, A. Giordana, L. Saitta, and M. Sebag. Relational learning: Hard problems and phase transitions. In *Proceedings of the Sixth Congress AI*AI*, volume 1792 of *LNAI*, pages 178–189, Berlin, 1999. Springer-Verlag.
5. H.C. Chen, C.H. Chen, L. Dai, and E. Yucesan. New Development of Optimal Computing Budget Allocation for Discrete Event Simulation. In K.J. Healy, D.H. Withers, and B.L. Nelson, editors, *Proceedings of the 1997 Winter Simulation Conference*, pages 334–341, 1997.
6. J.D. Gibbons, I. Olkin, and M. Sobel. *Selecting and Ordering Populations: A New Statistical Methodology*. John Wiley & Sons, New York, 1977.
7. S.S. Gupta and M. Sobel. Selecting a subset containing the best of several populations. In I. Olkin et al., editor, *Contributions to Probability and Statistics*, pages 224–248. Stanford University Press, Stanford, California, 1960.
8. W.J. Hall. The most economical character of Bechhofer and Sobel decision rules. *Annals of Mathematical Statistics*, 30:964–969, 1959.
9. Y.C. Ho. Heuristics, Rules of Thumb, and the 80/20 Proposition. *IEEE Transactions on Automatic Control*, 39(5):1025–1027, 1994.
10. T. M. Mitchell. *Machine Learning*. Mc-Graw-Hill, New York, 1997.
11. N. Lavrač, P. Flach, B. Zupan. Rule Evaluation Measures: A Unifying View. In S. Dzeroski and P.A. Flach, editors, *Proceedings of the Ninth International Workshop on Inductive Logic Programming (ILP99)*, volume 1634 of *LNAI*, pages 174–185, Berlin, 1999. Springer.
12. T. Scheffer and S. Wrobel. Finding the most interesting patterns in a database quickly by using sequential sampling. *Machine Learning Research*, 2002. To appear.
13. M. Sebag and C. Rouveirol. Tractable Induction and Classification in First-Order Logic via Stochastic Matching. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence (IJCAI-97)*. Morgan Kaufmann, Los Angeles, CA, 1997.
14. M. Sobel and M.J. Huyett. Selecting the best one of several binomial populations. *Bell System Technical Journal*, 36:537–576, 1957.
15. A. Srinivasan. A study of two sampling methods for analysing large datasets with ILP. *Data Mining and Knowledge Discovery*, 3(1):95–123, 1999.

16. L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.
17. W. Hoeffding. Probability inequalties for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

A An Upper Bound on the Probability of Wrong Selection

The main result here was derived before we were aware of the work on ranking and selection stemming from [2,14], or of the possibility of addressing the same question within the agnostic PAC setting. The main points of interest in the material below are that it: (i) illustrates by example, how probabilistic arguments for ranking (as opposed to estimation) can be constructed; (ii) shows that the probability of wrong selection falls away exponentially with increase in n and d. This is not immediately obvious from [14].

Consider two binomial populations $\Pi_{1,2}$ with parameters $p_{1,2}$. Let $p_{[1]}$ and $p_{[2]}$ denote ordered values of the p_i; and $X_{(i)}$ denote the (chance) number of successes that arises on n trials from the binomial process associated with $p_{[i]}$. Then, the probability of wrong selection P_{WS} with $p_{[1]} > p_{[2]}$ is given by:

$$
\begin{aligned}
P_{WS} &= P\{X_{(2)} > X_{(1)}\} + \frac{1}{2}P\{X_{(2)} = X_{(1)}\} \\
&\leq P\{X_{(2)} \geq X_{(1)}\} = \sum_{i=0}^{n} P\{X_{(2)} = i\}P\{X_{(1)} \leq i\} \\
&\leq \sum_{i=0}^{t} P\{X_{(2)} = i\}P\{X_{(1)} \leq i\} + \sum_{i=t+1}^{n} P\{X_{(2)} = i\}P\{X_{(1)} \leq i\} \quad (0 \leq t \leq n) \\
&\leq \sum_{i=0}^{t} P\{X_{(2)} = i\}P\{X_{(1)} \leq t\} + \sum_{i=t+1}^{n} P\{X_{(2)} \geq t\}P\{X_{(1)} \leq i\} \\
&\leq P\{X_{(1)} \leq t\}\sum_{i=0}^{t} P\{X_{(2)} = i\} + P\{X_{(2)} \geq t\}\sum_{i=t+1}^{n} P\{X_{(1)} \leq i\} \\
&\leq P\{X_{(1)} \leq t\} + P\{X_{(2)} \geq t\} \qquad (8)
\end{aligned}
$$

Since we simply require $0 \leq t \leq n$ we can chose $t = (m_1 + m_2)/2$ where $m_i = E\{X_{(i)}\} = np_{[i]}$. We are now in a position to use the following Hoeffding bounds [17] on the probability that the observed proportion of successes X/n differs from the expected proportion p by some amount ϵ $(0 \leq \epsilon \leq 1)$:

$$
\begin{aligned}
P\{X/n \geq p + \epsilon\} &\leq e^{-2n\epsilon^2} \\
P\{X/n \leq p - \epsilon\} &\leq e^{-2n\epsilon^2}
\end{aligned}
$$

With $\epsilon = (p_{[1]} - p_{[2]})/2$ these bounds in (8) yield:

$$
\begin{aligned}
P_{WS} &\leq 2e^{\frac{-2n(p_{[1]}-p_{[2]})^2}{4}} \\
&\leq 2e^{\frac{-nd^2}{2}} \qquad\qquad 0 < d = p_{[1]} - p_{[2]} \leq 1
\end{aligned}
$$

It is evident that as d increases, P_{WS} decreases. Suppose we are indifferent about wrong selection for values of $d \leq d^*$, then

$$P_{WS} \leq 2e^{\frac{-nd^{*2}}{2}} \tag{9}$$

where d^* $(0 < d^* \leq 1)$ is the smallest value of $p_{[1]} - p_{[2]}$ below which we are indifferent about a wrong selection. It is easy to extend this to selecting the best amongst k populations, for a wrong selection results if we select incorrectly on any of the $k-1$ comparisons of the $p_{[i]}(2 \leq i \leq k)$ against the best population $p_{[1]}$. An upper bound on the probability of each of these is given by (9) and the overall probability of wrong selection (with some abuse of notation) is bound by:

$$\begin{aligned} P_{WS} &\leq (k-1)2e^{\frac{-nd^{*2}}{2}} \\ &\leq 2ke^{\frac{-nd^{*2}}{2}} \end{aligned} \tag{10}$$

It is straightforward to relate this result to the probability of correct selection P_{CS} as used in the main body of this paper:

$$\begin{aligned} P_{CS} &= 1 - P_{WS} \\ &\geq 1 - 2ke^{\frac{-nd^{*2}}{2}} \end{aligned}$$

In particular, if we require $P_{CS} \geq P^*$ then:

$$n \geq \frac{2 \ln\left(\frac{2k}{1-P^*}\right)}{d^{*2}}$$

Compact Representation of Knowledge Bases in ILP

Jan Struyf, Jan Ramon, and Hendrik Blockeel

Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A, B-3001 Leuven, Belgium
{Jan.Struyf,Jan.Ramon,Hendrik.Blockeel}@cs.kuleuven.ac.be

Abstract. Many inductive systems, including ILP systems, learn from a knowledge base that is structured around examples. In practical situations this example-centered representation can cause a lot of redundancy. For instance, when learning from episodes (e.g. from games), the knowledge base contains consecutive states of a world. Each state is usually described completely even though consecutive states may differ only slightly. Similar redundancies occur when the knowledge base stores examples that share common structures (e.g. when representing complex objects as machines or molecules). These two types of redundancies can place a heavy burden on memory resources. In this paper we propose a method for representing knowledge bases in a more efficient way. This is accomplished by building a graph that implicitly defines examples in terms of other structures. We evaluate our method in the context of learning a Go heuristic.

1 Introduction

Inductive logic programming is concerned with the induction of new knowledge (hypotheses) from a given knowledge base. This knowledge base is in practice often structured around the notion of examples. The aim of this paper is to find a compact representation for these examples. We start by listing some motivating applications.

1. The task in relational reinforcement learning [7] is to learn a relationship between the structural description of a state and the optimal action for that state. A typical knowledge base will contain a number of episodes, sequences of states in which each state can be reached from the previous one by taking a certain action. Storing each state independently in the knowledge base consumes a lot of memory. An alternative is to store only the initial state together with a sequence of actions. Consecutive states can be constructed by the ILP system when needed (by executing the corresponding action from the previous state) and discarded afterwards.
2. When learning to play a certain game (e.g. Chess or Go [18]), the knowledge base will consist of a number of played games (generated by an expert human

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 254–269, 2003.

player or by a search algorithm). Each game is represented by a number of (consecutive) board states and moves. The same compact representation as described in the previous example can be used.

3. In some applications the examples are complex objects that can be decomposed in elementary structures. For instance machines built of various parts, state descriptions of a dungeon game where one can meet several monsters and own different types of weapons, or molecules that contain several functional groups. Representing examples independently in these cases will cause redundancy. This redundancy can be avoided by representing examples implicitly as a combination of their elementary structures.
4. Many tasks consist of classifying an individual element in a sequence based on its local context. Consider for instance protein secondary structure prediction [17], part of speech tagging [5] or user modeling [14]. Consecutive examples in this kind of knowledge bases are similar because the neighborhood of adjacent positions is similar. A compact representation is possible by defining a window for each example instead of explicitly storing the neighborhood.

As can be seen from the examples above, we need a formalism to represent individual examples in terms of previous examples (i.e. by using actions or, more generally, difference information) and in terms of elementary structures. We will introduce such a formalism in this paper.

Note that it is also possible to represent the set of examples more compactly in an ILP system by defining background knowledge B. For example, one could define different machine parts in B or one could represent state information implicitly using predicates defined in B. However, using this approach B must be redefined for each specific application. In this paper we propose a more general approach.

Using background knowledge for defining examples has two other disadvantages. (1) The ILP system must be able to load B in main memory. If we use B to store parts of the example descriptions (e.g. machine parts), then this puts a bound on the number of examples that the ILP system can use. (2) When using implicit state descriptions, the time for accessing a single example (state s_i) will depend on the number of previous states (we have to iterate over all previous states to obtain the definition of s_i). This means that the computational complexity of querying all examples (in random order) will be at least quadratic in the number of examples.

Scalability to large knowledge bases is obtained in practical ILP systems either (1) by connecting the ILP system to an external database management system (DBMS) [1,15,13] or (2) by using independent examples that can be loaded one by one, i.e. the Learning from Interpretations setting [3] in which each example is defined as a small relational database (typically implemented as a set of facts, i.e. an interpretation). The benefit of (2) over (1) is that (1) must rely for efficiency on the caching mechanism provided by the DBMS which lacks information about the specific way in which the ILP system will access the examples. In (2) this information is made explicit by combining all data relevant to a single example in an interpretation. This makes retrieving a single example

very efficient. The drawback of (2) is that examples have to be independent which means that compact representation (by defining examples in terms of other structures) is impossible. The framework presented in this paper extends the Learning from Interpretations setting by allowing dependent examples without sacrificing scalability.

The remainder of the paper is structured as follows. In Section 2 we introduce a formalism (the *knowledge base graph* or KBG) which enables us to represent examples in terms of other structures. In Section 3 we illustrate how the KBG can be described in Prolog. We discuss how an ILP algorithm can query the examples defined by the KBG efficiently in Section 4. Section 5 presents preliminary experiments evaluating our method in the context of learning a Go heuristic. In Section 6 we state the conclusions.

2 Knowledge Base Graph

In this section we propose the knowledge base graph or KBG as a formalism for representing examples in terms of other structures. We start by defining the KBG formally. After that we show that the KBG can be used to compactly represent each type of knowledge base mentioned in the introduction.

2.1 Hypergraphs

As the KBG is in fact a directed hypergraph, we would like to first briefly review directed hypergraphs. Directed hypergraphs are a generalization of normal graphs because they contain hyperedges which are able to express many-to-one relations (normal graph edges only relate two nodes).

Definition 1. *A directed hypergraph $\mathcal{H}$ is a pair $(N, \mathcal{E})$ where N is a set of nodes and $\mathcal{E}$ is a set of hyperedges. Each hyperedge ϵ is a tuple $(n_1, \ldots, n_k, n)$ from a set $\text{source}(\epsilon) = \{n_1, \ldots, n_k\} \subseteq N$ (source set) to a single node $n \in N$ (target node).*

Hypergraphs (See [11] for a discussion) are widely used in operations research and computer science (e.g. to model functional dependencies in relational databases, to model assembly plans and to represent Horn clauses, implications and and-or graphs). Hypergraphs are also used in knowledge representation [8] and [19] uses the hypergraph representation to parallelize a Knowledge Discovery System that embeds an ILP engine.

2.2 Structure

The KBG is a directed hypergraph with two sets of nodes: explicit nodes N_E and implicit nodes N_I. The explicit nodes store information from some domain D. We refer to the elements of D as "elementary structures" or "objects". In practice, elementary structures can be sets of facts, partial logic programs (sequences of clauses), terms [10], or application specific objects (Section 5).

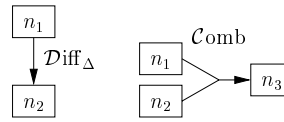


Fig. 1. Graphical representation of $\mathcal{D}$iff and $\mathcal{C}$omb hyperedges.

The implicit nodes do not store objects. However it is possible to compute an object associated with each implicit node. Computing this object is called explicitating the node. More formally, we define a function e*xpl* (explicitate) that maps each node to an object $o \in D$. For explicit nodes e*xpl* retrieves the object stored in the node. For implicit nodes, e*xpl* is defined (implicitly) by the hyperedges of the KBG.

Definition 2. *A difference hyperedge* $\mathcal{D}$iff *is a tuple* $\mathcal{D}\text{iff}_\Delta(n_1, n_2)$ *with* $n_1 \in N_E \cup N_I$ *and* $n_2 \in N_I$. *We have* $\mathcal{D}\text{iff}_\Delta(n_1, n_2)$ iff e$xpl(n_2)$ *can be derived from* e$xpl(n_1)$ *by applying difference information* Δ.

Each difference hyperedge $\mathcal{D}$iff stores some information Δ that describes how e$xpl(n_2)$ can be derived from e$xpl(n_1)$. In the context of games for example, Δ could state that a particular piece is placed on the board. We represent a difference hyperedge graphically as shown in Figure 1.

Definition 3. *A combine hyperedge* $\mathcal{C}$omb *is a tuple* $\mathcal{C}\text{omb}(n_1, n_2, n_3)$ *with* $n_1, n_2 \in N_E \cup N_I$ *and* $n_3 \in N_I$. *We have* $\mathcal{C}\text{omb}(n_1, n_2, n_3)$ iff e$xpl(n_3)$ *can be obtained by combining* e$xpl(n_1)$ *and* e$xpl(n_2)$.

The effect of combining e$xpl(n_1)$ and e$xpl(n_2)$ has to be defined for each specific choice of D. For example, if the objects are sets of facts, then the natural implementation for combining objects would be to take the union of these sets. We represent $\mathcal{C}$omb hyperedges graphically as shown in Figure 1.

In this paper we consider only $\mathcal{D}$iff and $\mathcal{C}$omb hyperedges. However, the KBG can be extended by defining more types of hyperedges. Note that $\mathcal{C}$omb hyperedges can also be extended to hyperedges that combine more than two nodes.

Definition 4. *The example nodes* $E_N \subseteq N_E \cup N_I$ *are the nodes of interest to the ILP system. The examples* E_o *are the objects associated with the example nodes* ($E_o = \{\text{e}xpl(n) \mid n \in E_N\}$).

2.3 Examples

In this section we show how some example knowledge bases can be represented using a KBG.

1. The Learning from Interpretations setting corresponds to the trivial form of the KBG when $N_E = E_N$, $N_I = \emptyset$, $\mathcal{E} = \emptyset$. In this case all examples are represented explicitly (i.e. there is no sharing).

2. Background knowledge B can be viewed as a node n_B that is combined with example descriptions $ed_i \in ED$ (e.g. explicit nodes that store sets of facts) to form example nodes $e_i \in E_N$ ($N_E = \{n_B\} \cup ED$, $N_I = E_N$, $\mathcal{E} = \{\mathcal{C}\text{omb}(ed_i, n_B, e_i) \mid ed_i \in ED,\ e_i \in E_N\}$).
3. When learning from episodes (e.g. in game play), we have one initial state for each episode i. Initial states $s_{i,0}$ are represented in the KBG as explicit nodes ($expl(n_{i,0}) = s_{i,0},\ n_{i,0} \in N_E$). Consecutive states $s_{i,j}$ are not stored explicitly and hence correspond to implicit nodes ($expl(n_{i,j}) = s_{i,j},\ n_{i,j} \in N_I$). $\mathcal{D}$iff hyperedges are used to describe the difference between $s_{i,j}$ and $s_{i,j+1}$. In this setting, the KBG is a set of sequences.
4. Sometimes a state can have several successors. This is useful for representing a number of *possible* actions in a state. In the context of games, a game subtree can be represented in this way. This is useful for learning from alpha-beta search trees (Section 5) or from games that allow variations. In this setting, the KBG is a set of trees.
5. It is also possible that some states can be reached with different sequences of moves/actions. If this happens, then the KBG will be a directed acyclic graph.
6. If examples share common substructures then we can use $\mathcal{C}$omb hyperedges. For the machines example it seems natural to have one (explicit) node for each part. Each example corresponds to an implicit node (a machine) that can be obtained by combining the different parts. Knowledge bases that store molecules or sequences can be represented in a similar way.
7. $\mathcal{C}$omb hyperedges can also be used in combination with $\mathcal{D}$iff hyperedges. Consider again the learning from game play example. Suppose that the examples for the ILP algorithm are the legal moves in each state. In this case the example nodes are implicit move-nodes that can be obtained by combining explicit move information (e.g. coordinates and value of the move) with the corresponding implicit state-node.

Note that when all possible actions are computable from a state (as is the case with most board games), the hyperedges and the implicit nodes of the KBG could be computed lazily. In that case the set of all possible games can be represented compactly by defining the initial state and a function mapping states to possible actions. The difference between a finite database of games and a world in which the learner can experiment becomes entirely transparent, offering an elegant method for active learning and random sampling.

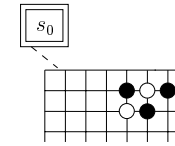
3 Representing the KBG in Prolog

In this section we illustrate how the KBG can be described in Prolog for the case where each object is a partial logic program (PLP). We will use Go [18] as example application. Go, a game popular in Asia is an abstract two-person complete-information deterministic board game (like Chess and Draughts).

3.1 Defining Explicit Nodes

Each explicit node stores a PLP (in our example the definition of a Go board state) and can be defined by placing the PLP between the keywords `begin_explicit_node(n)` and `end_explicit_node(n)` (Explicit nodes are represented graphically with a double rectangle).

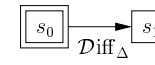
```
begin_explicit_node(s0).
      stone((2,5), black).
      stone((2,7), black).
      stone((3,6), black).
      stone((2,6), white).
      stone((3,5), white).
end_explicit_node(s0).
```



3.2 Defining $\mathcal{D}$iff Hyperedges

We define $\mathcal{D}$iff hyperedges using `diff_edge/3`. The first argument is the source node, the second argument the target node and the last argument specifies Δ. We can distinguish two abstraction levels for defining Δ. The first abstraction level is directly describing the difference between objects. In this case Δ specifies transformations on the PLP.

```
diff_edge(s0, s1, [+stone((1,6), black),
                   -stone((2,6), white)]).
```



In the example above Δ describes which clauses have to be added to (+) or removed from (-) the PLP. This way of representing $\mathcal{D}$iff hyperedges is very similar to the way operators are defined for a STRIPS [9] planner.

The second abstraction level is defining Δ using actions. In the Go example, this corresponds to storing moves instead of explicit object differences.

```
move(s0, s1, black, (1,6)).
...
diff_edge(S1, S2, Delta) :-
        move(S1, S2, Color, Coord),
        calculate_move_effect(Color, Coord, Delta).
```

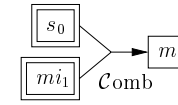
Note that `calculate_move_effect/3` will in most cases depend on the current state (i.e. the PLP associated with `S1`). This means that the knowledge base has to explicitate `S1` before using the `diff_edge/3` definition.

3.3 Defining $\mathcal{C}$omb Hyperedges

$\mathcal{C}$omb hyperedges can be defined using `comb_edge/3`. The semantics of a $\mathcal{C}$omb-(n_1, n_2, n_3) hyperedge are that the definitions of the PLP's associated with n_1 and n_2 are concatenated. In the Go example below, a move-node is defined by combining move information (e.g. the player who made the move, the move's coordinates and evaluation score) with a state-node.

```
begin_explicit_node(mi1).
      player(black).
      coordinate(1,6).
      score(1.0).
end_explicit_node(mi1).

comb_edge(mi1, s0, m1).
```



3.4 Defining Examples

The example nodes (in this case the moves) are a subset of the nodes of the KBG (Definition 4). We define this subset using `example/1`.

```
example(m1).
example(m2).
...
```

4 Traversing the KBG

In order to be practically useful, a KBG-based knowledge base should satisfy two key properties. The first one is transparency: the interface for interacting with the knowledge base should be the same as when all examples are represented explicitly. Transparency implies that existing ILP systems can be adapted easily to use a KBG-based knowledge base. The second property is computational efficiency: some loss of efficiency when using the KBG may be unavoidable, but it should remain limited (e.g. querying all examples should be $\mathcal{O}(N)$ with N the number of nodes in the KBG).

4.1 ILP Algorithm Interface

We call the interface component between the ILP algorithm and the KBG the *example iterator*. The task of the example iterator is to traverse the KBG and explicitate the example nodes that the ILP algorithm needs to query. Most ILP algorithms consider different subsets S of the available examples without imposing an order on the elements of S. This means that the example iterator can choose an order for iterating over S. Consider for example the KBG shown in Figure 2. The example iterator starts with the explicit node e_1. After that, it can move to e_2, explicitate e_2 by applying $\Delta 2$ and proceed with e_3. This means that the sequence e_1, e_2, e_3 can be iterated efficiently. A sequence like e_3, e_2, e_1 is less efficient because it is impossible to explicitate e_3 without constructing $expl(e_2)$ first. In Section 4.2 we will introduce the notion of this order in a more formal way. We conclude this section with some remarks on how different ILP algorithms query examples.

1. Some ILP algorithms build a model by considering different subsets of the available examples (e.g. Progol [16], Tilde [2]). Other algorithms perform different scans over the entire knowledge base (e.g. Warmr [6]). Algorithms

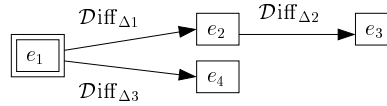


Fig. 2. An example KBG structure.

that perform full scans will execute more efficiently in combination with the KBG. Consider for example Figure 2 and suppose that the algorithm partitions the examples in $S_1 = \{e_1, e_3\}$, $S_2 = \{e_2, e_4\}$. In this case $expl(e_2)$ has to be constructed twice: once as intermediate result for $\{e_1, e_3\}$ and once for $\{e_2, e_4\}$. If the algorithm performs scans over the entire set $\{e_1, e_2, e_3, e_4\}$ then $expl(e_2)$ has to be constructed only once.

2. Most ILP algorithms evaluate a set of queries Q on a set of examples S. This can be implemented in two ways: execute each query on all examples (*queries outer loop*) or execute all queries on each example (*examples outer loop*). When used in combination with a KBG examples outer loop algorithms are more efficient. This is because the examples have to be constructed only once instead of once for each query. Note that full scan algorithms typically implement the examples outer loop setting.
3. Incremental ILP algorithms can use an infinite KBG. They access each example only once. This is also true for reinforcement learning [7]. Most other algorithms need a finite set of examples or work with different finite batches of examples.

4.2 A Planning Problem

The task of the example iterator is to construct each example $e \in S$ so that the ILP algorithm can query it. The example iterator must traverse the KBG in a certain order to accomplish this (consider again Figure 2). In the most general case, finding this order corresponds to solving a planning problem. Before we define this planning problem, we introduce the notion of active nodes.

A node n for which $expl(n)$ is in the main memory of the example iterator is called an active node (cached node). The example iterator can activate a node depending on its type.

1. An explicit node n can always be activated (by retrieving the object stored in n, i.e. loading $expl(n)$ from disk[1]).
2. An implicit node n can only be activated if there exists a hyperedge ϵ with n as target node and for which all nodes in the source set are active. Activating the node corresponds to constructing $expl(n)$ as defined by ϵ.

The reverse action, deactivating n, corresponds to removing $expl(n)$ from the main memory.

[1] This functionality could be provided by an object oriented DBMS.

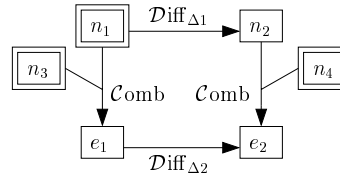


Fig. 3. A Planning Problem.

While deactivating a node can be seen as effortless, activating a node n involves a certain cost. For explicit nodes this is the cost of loading $expl(n)$ from disk and for implicit nodes it is the cost of constructing $expl(n)$. The example iterator should minimize this cost while performing its task. The number of active nodes at any given time should also be limited. This is because the main memory of the example iterator can only store a finite number of objects. More formally, it must hold that $|N_A| \leq max$ with N_A the set of active nodes and max the maximum number of objects that can be stored in memory. Maintaining memory usage can be done by deactivating nodes that are no longer necessary.

We now define the planning problem faced by the example iterator.

Definition 5. *(Planning problem faced by the example iterator) Given a set of active nodes $N_A = N_A(0)$ and a set of examples S that the ILP algorithm needs to query, find a sequence of actions a_i (activating or deactivating a node n_i) such that:*

1. *(Objective) $\sum_{a_i} cost(a_i)$ is minimal.*
2. *(Memory constraint) $\forall i,\ |N_A(i)| \leq max$.*
3. *(ILP algorithm constraint) $\forall e \in S,\ \exists i$ such that $e \in N_A(i)$.*

with $cost(a_i)$ the cost of performing a_i as defined above and $N_A(i)$ the set of active nodes after performing a_i.

Example 1. Consider the KBG shown in Figure 3. Suppose that the ILP algorithm needs to query the examples $S = \{e_1, e_2\}$ and that $max = 3$. The sequence $\{activate(n_1)$, $activate(n_3)$, $activate(e_1)$, $deactivate(n_3)$, $activate(e_2)\}$ is a plan that satisfies both the memory and the ILP algorithm constraint. Whether or not this plan is optimal depends on the costs associated with the actions. For example if $\mathcal{D}\text{iff}_{\Delta 2}$ implies a high cost, then $\{activate(n_1)$, $activate(n_3)$, $activate(e_1)$, $deactivate(n_3)$, $activate(n_2)$, $deactivate(n_1)$,$deactivate(e_2)\}$ may be a cheaper solution.

The planning problem faced by the example iterator is in general NP-hard. This can be seen by considering a special instance that is equivalent with the Traveling Salesman Problem (TSP). Suppose that the KBG has one explicit node, a number of implicit nodes and that all hyperedges are $\mathcal{D}$iff's. All nodes are examples for the ILP algorithm. If $max = 2$ then traversing $\mathcal{D}\text{iff}_{\Delta}(n_1, n_2)$ implies

first activating n_2 and afterwards deactivating n_1. In this case the planning problem is reduced to that of finding a minimal cost path through the KBG while visiting all nodes once. This is a variant of TSP.

4.3 An Efficient Example Iterator

In practice we need an algorithm that scales well in the number of nodes of the KBG. For some specific structures of the KBG it is possible to solve the planning problem efficiently. For example, if the KBG is a set of sequences then the planning problem becomes trivial. For other structures it is more difficult. We propose an efficient algorithm that approximates the planning problem for a subset of all possible KBG structures which we call *restricted* KBG structures ($RKBG$).

Definition 6. *(Restricted KBG structure). Let $T \in \mathbb{R}^+$ be a constant. If it holds for a certain KBG structure and for an increasing number of nodes N that:*

1. *Activating an explicit node can be done in time $\leq T$.*
2. *Activating an implicit node n can be done in time $\leq T$ if $\exists \epsilon \in \mathcal{E}$ that defines n for which $\text{source}(\epsilon) \subseteq N_A$.*
3. *For each $\mathcal{C}\text{omb}(n_1, n_2, n_3)$ hyperedge either $\text{expl}(n_1)$ or $\text{expl}(n_2)$ can be computed (using the algorithm shown in Figure 4) in time $\leq T$. We call the corresponding node (either n_1 or n_2) the efficient node of the $\mathcal{C}$omb hyperedge.*

then we call this KBG structure a restricted KBG structure.

Definition 7. *(Forest). A KBG is called a forest if each node n is the target of at most one hyperedge.*

A KBG structure that satisfies Definition 6 can be iterated in linear $\mathcal{O}(N)$ time, with N the number of nodes. One can implement such a linear time example iterator as follows.

1. Compute the spanning forest (Definition 7) of the KBG. This can be done in $\mathcal{O}(N)$. The KBG of Figure 3 can for example be transformed into a forest by removing $\mathcal{D}\text{iff}_{\Delta 2}$. A better approximation of the planning problem is obtained by calculating the *minimal* spanning forest. This can be done using an adapted version of Prim-Jarník's [12] algorithm in $\mathcal{O}(E\ log\ N)$, with E the number of hyperedges and N the number of nodes of the KBG. Computing the spanning forest should be done only once when the knowledge base is initialized.
2. Find the root set R of the spanning forest. The root set is the set of all explicit nodes that are not ancestors of efficient nodes (See Definition 6). The root set of the KBG shown in Figure 3 is $R = \{n_1\}$. Explicit nodes n_3 and n_4 are not in the root set because they are efficient nodes.
3. Repeat steps 4-6 for each root $r \in R$.

```
function explicitate(n)
    if n ∈ N_A then return expl(n)
    else if n ∈ N_E then
        return load_from_disk(n)
    else if ∃n_1 : Diff_Δ(n_1, n) then
        o1 = explicitate(n_1); o2 = apply(Δ, o_1)
        deactivate(n_1); return o2
    else if ∃n_1, n_2 : Comb(n_1, n_2, n) then
        o1 = explicitate(n_1); o2 = explicitate(n_2); o3 = combine(o1, o2)
        deactivate(n_1); deactivate(n_2); return o3
    else raise illegal_kbg_structure
```

Fig. 4. An algorithm for constructing $expl(n)$.

4. Activate r and traverse the tree with root r depth-first. Traversing a hyperedge means activating the corresponding child node (See steps 5 and 6). A node can be deactivated after all of its children have been visited.
5. Traversing a $\mathcal{D}\text{iff}_{\Delta}(n_1, n_2)$ hyperedge is trivial: construct $expl(n_2)$ by applying Δ to $expl(n_1)$.
6. Traversing a $\mathcal{C}\text{omb}(n_1, n_2, n_3)$ hyperedge is more difficult. Suppose that n_2 is its efficient node. Before we can activate n_3, we must compute $expl(n_2)$. Because of Definition 6, we know that this can be done efficiently (using the algorithm shown in Figure 4). After n_2 is activated we can construct $expl(n_3)$ by combining $expl(n_1)$ and $expl(n_2)$. Afterwards, n_2 can be deactivated.

Example 2. Consider again the KBG shown in Figure 3. Removing $\mathcal{D}\text{iff}_{\Delta 2}$ results in the root set $R = \{n_1\}$. We now illustrate steps 4-6. First, we activate n_1 by loading it from disk. Root n_1 has two children: n_2 and e_1. Suppose that the algorithm starts by activating n_2. The next step is traversing the $\mathcal{C}$omb hyperedge leading to e_2. This is trivial because n_4 is an explicit node. Node n_2 can be deactivated because it has only one child. e_2 can also be deactivated (after passing $expl(e_2)$ to the ILP algorithm). Root n_1 has one child left: e_1. Activating this child is analogous to activating e_2. The last step of the algorithm is deactivating n_1 after all its children have been processed.

We end this section with some remarks about our algorithm.

1. The number of nodes that can be active at a given time is bounded by the maximal tree depth d. In most practical cases d is small compared to the number of nodes. Note that we should only keep internal nodes active that have more than one successor: for sequences, the algorithm deactivates each node immediately. If the number of active nodes grows too large (the objects do not fit in memory), then the algorithm should deactivate some nodes. This implies that some objects have to be reconstructed which can increase the computational complexity of the algorithm.
2. If the efficient nodes of two $\mathcal{C}$omb hyperedges share some ancestors, then these ancestors are constructed several times. If enough memory is available,

it may be better to cache these ancestors (i.e. keep them active). It can also be useful to keep explicit nodes active in order to avoid loading them multiple times.

3. Sometimes the hyperedges of the KBG can be traversed in the opposite direction. For a $\mathcal{D}$iff hyperedge this means undoing Δ. In this case it is not necessary to keep nodes with more than one child active until all children are processed. This method can be combined with destructive updating the objects. For some applications it is more efficient to change an existing object than to construct a new object. For example in Go it is much more efficient to put one stone on an existing board than constructing a new board with the extra stone. Destructive updates can only be done if the source node may be deactivated after traversing a given hyperedge.
4. If the ILP algorithm needs to query small subsets S then it is possible that some parts of the KBG are irrelevant. Consider for example Figure 3 and suppose that $S = \{e_1\}$. This implies that e_2, n_4 and n_2 are not needed anymore. An obvious optimization to the proposed algorithm is to start with a step that marks all nodes (indirectly) needed by the ILP algorithm (in the example e_1, n_3 and n_1). In step 4 it is then enough to consider trees with marked roots. When traversing these trees, unmarked children do not need to be processed.

5 Experimental Evaluation

In order to evaluate our method we did some preliminary experiments on an application in the context of Go [18]. Unlike other two-person complete-information deterministic games such as Chess and Draughts, brute force search is not sufficient to let computers play Go at a level comparable to human players. Therefore, research is necessary on more intelligent methods. One of such is to learn heuristics in order to solve local problems more efficiently. In [18] we applied the Tilde [2,4] system (a first order decision tree induction system) to a set of local problems to learn a heuristic that can be used in an alpha-beta search to order moves and hence improve pruning of bad moves. In [18] we only predicted the first move that should be done in a problem. Further research now focuses on integrating the Tilde system and the search algorithm to learn better heuristics. The search algorithm first solves a problem, and all the nodes it visits are logged. Afterwards, heuristics are learned to optimize the search process.

From the log generated by the search algorithm we can construct a KBG as shown in Figure 5. The backbone of this KBG is formed by the states visited by the alpha-beta search (state-nodes s_i connected with $\mathcal{D}$iff hyperedges).

The search algorithm uses an iterative deepening method. In each iteration a number of states are expanded further, i.e. more nodes of the subtree under that node are investigated. Hence each state-node of the tree is expanded one or more times. A *node_expand* node in the KBG can be constructed by combining a state-node with a *node_expand_info* node. This *node_expand_info* node contains information specific to a particular visit of the algorithm to the corresponding

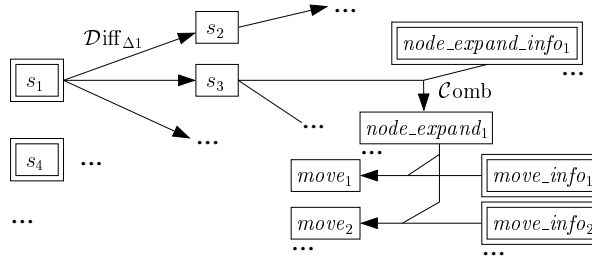


Fig. 5. The KBG for the Go application.

state-node, such as the amount of time spent and the depth to which this node was expanded. In each visit a set of candidate moves is investigated. This introduces a third kind of implicit nodes in the KBG: the move-nodes. A move-node is defined by combining a *node_expand* node and a *move_info* node that contains the coordinates of the move, the value of the move that was calculated during this particular node expansion, a measure for the difficulty to find an accurate estimate of the value of the move, etc.

We have implemented a KBG in the knowledge base of the ILP system ACE [4] (which includes Tilde). For the Go application, the example iterator does not cache objects at internal nodes of the KBG trees. Instead it uses destructive updating of the Go states as discussed in Section 4.3. The example iterator loads all objects associated with explicit nodes in main memory during initialization. Our current implementation does not have a caching strategy for on-demand loading. It does not include support for manipulating PLP's either. Instead, we use an application specific library[2] written in C++ designed to efficiently represent and manipulate Go states.

Table 1 shows a comparison between the compact (i.e. using the KBG described above) and full (all board states are represented explicitly[3]) representation of the Go knowledge base. Each row corresponds to a single experiment for which the knowledge base was obtained by changing parameters (max number of nodes to visit, minimal depth of the search that is logged) of the search algorithm. The first column contains the number of moves (i.e. examples). The other columns contain the sizes of the knowledge base on disk and the runtime of the Tilde system. We measured runtimes using the Query packs [4] and the *queries outer loop* setting of Tilde. Query packs is an optimized version of *examples outer loop* (Section 4.1).

When using the Query pack[4] mechanism, the differences in runtime between the experiments on the compact and full representation are minimal (even diffi-

[2] Available for academic purposes upon request.

[3] If more than one move is explored for a given board state, then this state is not replicated for each move.

[4] The Query pack experiments run efficient because the Query packs have a long root and a high branching factor in this application. Query packs are also pre-compiled.

Table 1. Comparison between the compact and full representation of the Go knowledge base. We report the size (in Kb) of the knowledge base on disk and the runtime (in seconds) of the TILDE system (Query packs [4] and *queries outer loop*).

# moves	Size			Query packs			Queries outer loop		
	Full	Compact		Full	Compact		Full	Compact	
1269	634	153	4.1×	11.9	12.1	-2%	864	840	3%
2795	1580	369	4.3×	17.9	17.5	2%	1270	1250	2%
5669	4047	845	4.8×	52.9	52.7	0%	5180	5110	1%
9384	9668	1919	5.0×	28.2	28.1	0%	793	788	1%
14549	21776	4173	5.2×	59.6	59.6	0%	1200	1190	1%
172411	194098	37925	5.1×	288.0	272.0	6%	3840	3490	10%

cult to measure). We also tried the less efficient query outer loop setting (Section 4.1), where the ILP system iterates for each candidate query over all examples. As this approach needs more computational effort of the example iterator, the experiments on the compactly represented knowledge base are slightly slower. In the last row of the table, larger overhead occurs (6% resp. 10%). This overhead is introduced by the sampling method of TILDE (TILDE uses samples of 1000 moves at each node of the decision tree). If TILDE queries only a small subset of states in the KBG then the overhead will increase because intermediate states have to be constructed in the compact representation without being used.

On the other hand one can see that the compactly represented knowledge bases are a factor of magnitude smaller than the full knowledge bases. The reduction is about 4× for small experiments and increases to about 5× for larger ones. The factor increases because the KBG's of larger experiments have more implicit state-nodes (i.e. nodes of the search tree) compared to the number of stored initial states.

Our experiments demonstrate that it is possible to obtain a significant reduction in memory consumption at virtually no extra computational cost by using a KBG-based knowledge base for Go. For other applications, where states are more complex (in Go, board states are represented very efficiently using only 2 bits for each field), we expect larger reductions in memory consumption. Because we use an application specific library for manipulating Go states $\mathcal{D}$iff and $\mathcal{C}$omb operations are very efficient. If we use PLP's we can expect that these operations will be more complex, yielding a larger overhead. For large knowledge bases, the time necessary for loading objects may be larger than the time necessary for constructing them. If this is the case then it is possible that the runtime actually decreases when using a KBG [5].

6 Conclusions

In many applications, knowledge bases for ILP algorithms store redundant information. Typical examples are learning from episodes where consecutive states

[5] The time for loading the knowledge base was not included in the reported runtimes.

of a world have to be stored, and learning from objects composed from elementary structures. In this paper we introduced a formalism, the *knowledge base graph* (KBG), which is able to represent examples implicitly in terms of other structures. Using this formalism, it is possible to remove most of the redundancy from the knowledge base.

Most ILP systems can be adapted easily to work with a KBG. In order to accomplish this we introduced an interface between the ILP system and the knowledge base: the *example iterator*. The example iterator has to iterate the examples stored in the KBG in an optimal order. This corresponds to an NP-hard planning problem for which we propose an efficient approximation that can iterate the examples in linear time (for the case of a *restricted* KBG).

We evaluated our method in the context of learning a Go heuristic. For this application it is possible to obtain a significant reduction in memory consumption (up to $5\times$) at virtually no extra computational cost ($\leq 10\%$). We also briefly discussed how we expect these results to generalise to other applications.

Further work includes more experiments with other knowledge bases. The first thing we would like to investigate is knowledge bases that store partial logic programs (PLP's). In order to manipulate PLP's efficiently it would be good to have special support from the Prolog engine. Furthermore, we would like to look at knowledge bases that are lazily sampled from the whole example space. For some applications (e.g. knowledge bases storing molecules) it is less obvious how to decompose structured objects. For these types of applications, it would be interesting to have a system that could separate common substructures automatically. Such a system is perhaps more generally applicable for optimizing the compilation of a knowledge base.

Acknowledgments

Jan Struyf is a research assistant, Hendrik Blockeel a post-doctoral fellow of the Fund for Scientific Research (FWO) of Flanders. Jan Ramon is supported by the Flemish Institute for the Promotion of Science and Technological Research in Industry (IWT). The presentation of this paper at ILP'02 is supported by ILPnet2 (http://www.cs.bris.ac.uk/~ILPnet2/).

References

1. H. Blockeel and L. De Raedt. Relational knowledge discovery in databases. In *Proceedings of the Sixth International Workshop on Inductive Logic Programming*, volume 1314 of *Lecture Notes in Artificial Intelligence*, pages 199–212. Springer-Verlag, 1996.
2. H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, June 1998.
3. H. Blockeel, L. De Raedt, N. Jacobs, and B. Demoen. Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*, 3(1):59–93, 1999.

4. H. Blockeel, L. Dehaspe, B. Demoen, G. Janssens, J. Ramon, and H. Vandecasteele. Improving the efficiency of inductive logic programming through the use of query packs. *Journal of Artificial Intelligence Research*, 16:135–166, 2002.
5. J. Cussens. Part-of-speech tagging using progol. In *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, Lecture Notes in Artificial Intelligence, pages 93–108. Springer-Verlag, 1997.
6. L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.
7. S. Džeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning*, 43:7–52, 2001.
8. A. Fall and G. W. Mineau. Knowledge retrieval, use and storage for efficiency. *Computational Intelligence*, 15:1–10, 1999.
9. R.E. Fikes and N.J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189 – 208, 1971.
10. P.A. Flach. Strongly typed inductive concept learning. In D. Page, editor, *Proceedings of the Eighth International Conference on Inductive Logic Programming*, volume 1446, pages 185–194. Springer-Verlag, 1998.
11. G Gallo, G Longo, S Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42:177–201, 1993.
12. M. T. Goodrich and R. Tamassia. *Algorithm Design*. Wiley, 2002.
13. Masaki Ito and Hayato Ohwada. Efficient database access for implementing a scalable ILP engine. In *Work-In-Progress Report of the Eleventh International Conference on Inductive Logic Programming*, 2001.
14. N. Jacobs and H. Blockeel. From shell logs to shell scripts. In *Proceedings of ILP2001 - Eleventh International Workshop on Inductive Logic Programming*, volume 2157 of *Lecture Notes in Artificial Intelligence*, pages 80–90, 2001.
15. K. Morik and P. Brockhausen. A multistrategy approach to relational discovery in databases. *Machine Learning*, 27(3):287–312, 1997.
16. S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
17. S. Muggleton, R.D. King, and M.J.E. Sternberg. Protein secondary structure prediction using logic-based machine learning. *Protein Engineering*, 7:647–657, 1992.
18. J. Ramon, T. Francis, and H. Blockeel. Learning a Tsume-Go heuristic with Tilde. In *Proceedings of CG2000, the Second international Conference on Computers and Games*, volume 2063 of *Lecture Notes in Computer Science*, pages 151–169. Springer-Verlag, 2000.
19. J. Seitzer, J. P. Buckley, and Y. Pan. INDED: A distributed knowledge-based learning system. *IEEE Intelligent Systems*, 15(5):38–46, 2000.

A Polynomial Time Matching Algorithm of Structured Ordered Tree Patterns for Data Mining from Semistructured Data

Yusuke Suzuki[1], Kohtaro Inomae[1], Takayoshi Shoudai[1], Tetsuhiro Miyahara[2], and Tomoyuki Uchida[2]

[1] Department of Informatics, Kyushu University, Kasuga 816-8580, Japan
{y-suzuki,k-inomae,shoudai}@i.kyushu-u.ac.jp
[2] Faculty of Information Sciences
Hiroshima City University, Hiroshima 731-3194, Japan
{miyahara@its,uchida@cs}.hiroshima-cu.ac.jp

Abstract. Tree structured data such as HTML/XML files are represented by rooted trees with ordered children and edge labels. Knowledge representations for tree structured data are quite important to discover interesting features which such tree structured data have. In this paper, as a representation of structural features we propose a structured ordered tree pattern, called a term tree, which is a rooted tree pattern consisting of ordered children and structured variables. A variable in a term tree can be substituted by an arbitrary tree.
Deciding whether or not each given tree structured data has structural features is a core problem for data mining of large tree structured data. We consider a problem of deciding whether or not a term tree t matches a tree T, that is, T is obtained from t by substituting some trees for variables in t. Such a problem is called a membership problem for t and T. Given a term tree t and a tree T, we present an $O(nN)$ time algorithm of solving the membership problem for t and T, where n and N are the numbers of vertices in t and T, respectively. We also report some experiments on applying our matching algorithm to a collection of real Web documents.

1 Introduction

Backgrounds: In the fields of data mining and knowledge discovery, many researchers have developed techniques for extracting common characteristics among given data. Due to rapid growth of Information Technologies, the amount of electronic data, especially Web documents, has increased rapidly. Web documents such as HTML/XML files have tree structures and are called tree structured data or semistructured data. We represent such tree structured data by rooted trees with ordered children and edge labels, based on the Object Exchange Model [1]. As examples of representations of tree structured data, we give rooted trees T_1, T_2 and T_3 in Fig. 1.

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 270–284, 2003.

Main Results: First, as a representation of a tree structured pattern in such tree structured data, we propose a structured ordered tree pattern, called a *term tree*, which is a rooted tree pattern with ordered children consisting of tree structures and structured variables. Secondly, we consider a problem of deciding whether or not a term tree t matches a tree T, that is, T is obtained from t by substituting some trees for variables in t. In case of regarding a term tree as a knowledge representation, this problem is a core problem for data mining of tree structured data. For designing a data mining technique of extracting structural features from tree structured data, an efficient algorithm for solving the above problem for each term tree and each given tree structured data is needed. In this paper, when a term tree t and a tree T are given as input, we present an $O(nN)$ time algorithm for solving the problem for t and T, where n and N are the numbers of vertices in t and T, respectively.

Finally, in order to show efficiency of our algorithm for real Web documents, we report an implementation and experimental results of the algorithm on a collection of real Web documents.

Detailed Explanation of Main Results: A term tree is different from the other representations such as a tree-expression pattern [17] and a regular path expression [6] in that a term tree has structured variables which can be substituted by arbitrary trees and also a term tree represents not a substructure but a whole structure of a tree structured pattern. A term tree is more powerful than a standard tree pattern, which is also called a first order term in formal logic, in computational learning theory [2]. For example, in Fig. 1, the standard tree pattern $f(b, x, g(a, z), y)$ can be represented by the term tree s, but the term tree t' can not be represented by any standard tree pattern because of the existence of internal structured variables represented by x_2 and x_3 in t'. Further, since a variable in a term tree is defined as a list of vertices and can be substituted by an arbitrary tree, a term tree is more powerful than or incomparable to other representations of tree structured patterns, which were proposed in computational learning theory, such as ordered tree patterns [2,3] and ordered gapped tree patterns [5].

For a set of edge labels Λ, the *term tree language* of a term tree t, denoted by $L_\Lambda(t)$, is the set of all labeled trees which are obtained from t by substituting arbitrary labeled trees for all variables in t. A term tree t is said to be *regular* if all variable labels in t are mutually distinct.

In this paper, we deal with the set $\mathcal{OTT}^{*,*}_{\Lambda}$ of all regular term trees with Λ as a set of edge labels, where Λ is any finite or infinite set of edge labels. Let $\mathcal{R}$ be a set of regular term trees. The *membership problem* for $\mathcal{R}$ is, given a term tree $t \in \mathcal{R}$ and a tree T, to decide whether or not $T \in L_\Lambda(t)$. The first main result is to give a polynomial time algorithm for the membership problem for $\mathcal{OTT}^{*,*}_{\Lambda}$. A term tree t is said to *match* a tree T if T is in $L_\Lambda(t)$. So a membership algorithm for $\mathcal{R}$ is also called a matching algorithm for $\mathcal{R}$. Consider the examples of the term tree t' and the tree T_1 in Fig. 1. It holds that $L_\Lambda(t')$ includes T_1, because T_1 is obtained from t' by replacing variables represented by x_1, x_2 and x_3 in t' with the trees g_1, g_2 and g_3 in Fig. 1, respectively.

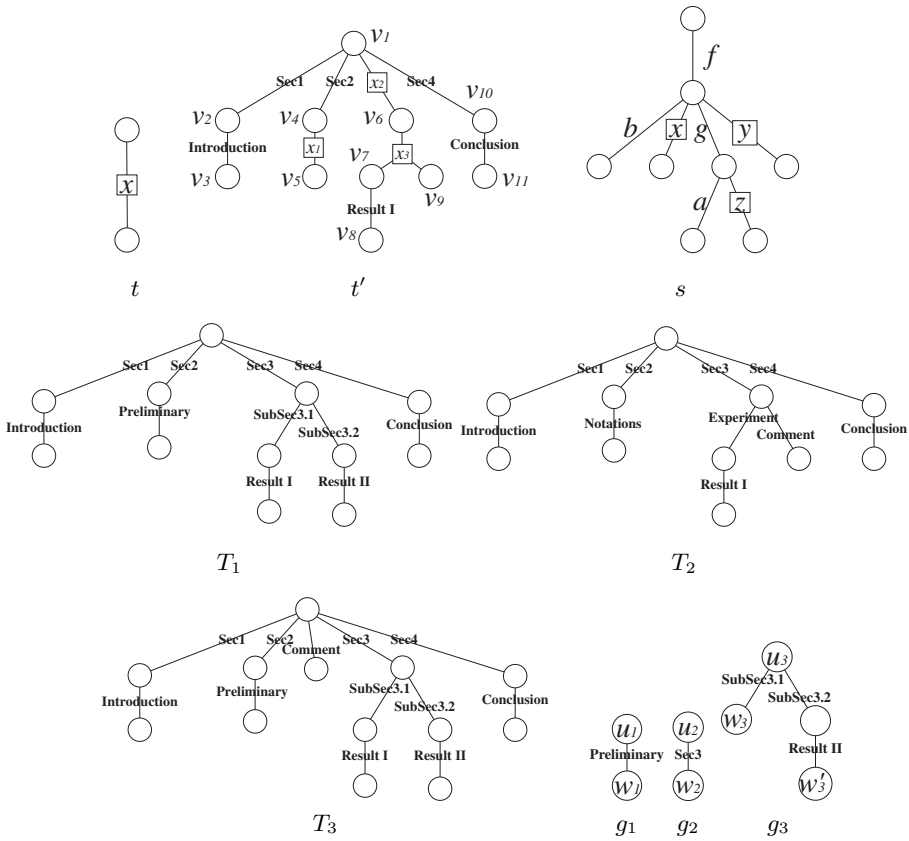


Fig. 1. Term trees t and t' match trees T_1, T_2 and T_3. A term tree s represents the standard tree pattern $f(b, x, g(a, z), y)$. A variable is represented by a box with lines to its elements. The label of a box is the variable label of the variable.

Related Works: A variable in a term tree is considered a list of vertices. For example, the variable represented by x_3 in a term tree t' in Fig. 1 is considered a list $[v_6, v_7, v_9]$. Let $L \geq 1$ and $K \geq 1$ be integers. Let $\mathcal{OTT}_{\Lambda}^{L,K}$ be the set of all regular term trees t with Λ as a set of edge labels such that each variable in t consists of at most $L + 1$ vertices and any path from the root to a leaf in t has at most K variables. Let $\mathcal{OTT}_{\Lambda}^{L,*} = \bigcup_{K \geq 1} \mathcal{OTT}_{\Lambda}^{L,K}$. The set $\mathcal{OTT}_{\Lambda}^{*,*}$ equals $\bigcup_{L \geq 1} \bigcup_{K \geq 1} \mathcal{OTT}_{\Lambda}^{L,K}$. Since a term tree in $\mathcal{OTT}_{\Lambda}^{*,*}$ has no restriction on the variables of it, the set $\mathcal{OTT}_{\Lambda}^{*,*}$ can represent general forms of ordered tree structured patterns. The *minimal language (MINL) problem* for a set of regular term trees $\mathcal{R}$ is, given a set of trees S, to find a term tree $t \in \mathcal{R}$ such that $L_{\Lambda}(t)$ is minimal among all languages $L_{\Lambda}(t')$ for $t' \in \mathcal{R}$ with $S \subseteq L_{\Lambda}(t')$.

As related works, in [7,13], we gave some sets of *unrooted* regular term trees with *unordered* children whose languages are polynomial time inductively infer-

able from positive data [4,12]. In [9,14], we considered the complexities of the membership problem and the MINL problem for a set of rooted regular term trees with unordered children. In [10,11], we gave data mining systems having special types of *rooted* regular term trees with unordered children and ordered children as knowledge representations, respectively. In [11], we considered a method for data mining of ordered term trees in $\mathcal{OTT}_{\Lambda}^{1,*}$ from *positive* semistructured data. From the view point of computational learning theory, we gave polynomial time algorithms for MINL for $\mathcal{OTT}_{\Lambda}^{1,*}$ and $\mathcal{OTT}_{\Lambda}^{1,1}$ in [15], and $\mathcal{OTT}_{\Lambda}^{*,*}$ in [16]. And any finite union of $\mathcal{OTT}_{\Lambda}^{1,*}$ was shown to be exactly identifiable in polynomial time by query learning model [8].

Organization: This paper is organized as follows. In Section 2, we propose term trees as tree structured patterns. In Section 3, we give a polynomial time membership algorithm for $\mathcal{OTT}_{\Lambda}^{*,*}$, where Λ is any finite or infinite set of edge labels. Further, in Section 4 in order to show efficiency of our algorithm, we report some experimental results of our algorithm on a collection of real Web pages.

2 Preliminaries – Ordered Term Trees

Let $T = (V_T, E_T)$ be an ordered tree with a vertex set V_T and an edge set E_T. A list $h = [u_0, u_1, \ldots, u_\ell]$ of vertices in V_T is called a *variable* of T if $u_1, \ldots, u_\ell$ is a sequence of consecutive children of u_0, i.e., u_0 is the parent of $u_1, \ldots, u_\ell$ and u_{j+1} is the next sibling of u_j for j with any $1 \leq j < \ell$. We call u_0 the *parent port* of the variable h and $u_1, \ldots, u_\ell$ the *child ports* of h. Two variables $h = [u_0, u_1, \ldots, u_\ell]$ and $h' = [u'_0, u'_1, \ldots, u'_{\ell'}]$ are said to be *disjoint* if $\{u_1, \ldots, u_\ell\} \cap \{u'_1, \ldots, u'_{\ell'}\} = \emptyset$.

Definition 1. Let $T = (V_T, E_T)$ be an ordered tree and H_T a set of pairwise disjoint variables of T. An *ordered term tree obtained from T and H_T* is a triplet $t = (V_t, E_t, H_t)$ where $V_t = V_T$, $E_t = E_T - \bigcup_{h=[u_0,u_1,\ldots,u_\ell] \in H_T} \{\{u_0, u_i\} \in E_T \mid 1 \leq i \leq \ell\}$, and $H_t = H_T$. For two vertices $u, u' \in V_t$, we say that u is the *parent* of u' in t if u is the parent of u' in T. Similarly we say that u' is a *child* of u in t if u' is a child of u in T. In particular, for a vertex $u \in V_t$ with no child, we call u a *leaf* of t. We define the order of the children of each vertex u in t as the order of the children of u in T. We often omit the description of the ordered tree T and variable set H_T because we can find them from the triplet $t = (V_t, E_t, H_t)$.

For example, the ordered term tree t' in Fig. 1 is obtained from the tree $T = (V_T, E_T)$ and the set of variables H_T defined as follows. $V_T = \{v_1, \ldots, v_{11}\}$, $E_T = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_4\}, \{v_4, v_5\}, \{v_1, v_6\}, \{v_6, v_7\}, \{v_7, v_8\}, \{v_6, v_9\}, \{v_1, v_{10}\}, \{v_{10}, v_{11}\}\}$ with the root v_1 and the sibling relation displayed in Fig. 1. $H_T = \{[v_4, v_5], [v_1, v_6], [v_6, v_7, v_9]\}$.

For any ordered term tree t, a vertex u of t, and two children u' and u'' of u, we write $u' <_u^t u''$ if u' is smaller than u'' in the order of the children of u. We assume that every edge and variable of an ordered term tree is labeled with some words from specified languages. A label of a variable is called a *variable label*. Λ

and X denote a set of edge labels and a set of variable labels, respectively, where $\Lambda \cap X = \phi$. An ordered term tree $t = (V_t, E_t, H_t)$ is called *regular* if all variables in H_t have mutually distinct variable labels in X.

Note. In this paper, we treat only regular ordered term trees, and then we call a regular ordered term tree a *term tree*, simply. In particular, an ordered term tree with no variable is called a *ground term tree* and considered to be a tree with ordered children.

For a term tree t and its vertices v_1 and v_i, a *path* from v_1 to v_i is a sequence $v_1, v_2, \ldots, v_i$ of distinct vertices of t such that for any j with any $1 \leq j < i$, v_j is the parent of v_{j+1}. $\mathcal{OT}_\Lambda$ denotes the set of all ground term trees with Λ as a set of edge labels. Let $L \geq 1$ and $K \geq 1$ be integers. Let $\mathcal{OTT}_\Lambda^{L,K}$ be the set of all term trees t with Λ as a set of edge labels such that each variable in t has at most L child ports and any path from the root to a leaf in t has at most K variables. Let $\mathcal{OTT}_\Lambda^{*,*} = \bigcup_{L\geq 1} \bigcup_{K \geq 1} \mathcal{OTT}_\Lambda^{L,K}$.

Let $f = (V_f, E_f, H_f)$ and $g = (V_g, E_g, H_g)$ be term trees. We say that f and g are *isomorphic*, denoted by $f \equiv g$, if there is a bijection φ from V_f to V_g such that (i) the root of f is mapped to the root of g by φ, (ii) $\{u, u'\} \in E_f$ if and only if $\{\varphi(u), \varphi(u')\} \in E_g$ and the two edges have the same edge label, (iii) $[u_0, u_1, \ldots, u_\ell] \in H_f$ if and only if $[\varphi(u_0), \varphi(u_1), \ldots, \varphi(u_\ell)] \in H_g$, and (iv) for any vertex u in f which has more than one child, and for any two children u' and u'' of u, $u' <_u^f u''$ if and only if $\varphi(u') <_{\varphi(u)}^g \varphi(u'')$.

Let f and g be term trees with at least two vertices. Let $h = [v_0, v_1, \ldots, v_\ell]$ be a variable in f with the variable label x and $\sigma = [u_0, u_1, \ldots, u_\ell]$ a list of $\ell + 1$ distinct vertices in g where u is the root of g and $u_1, \ldots, u_\ell$ are leaves of g. The form $x := [g, \sigma]$ is called a *binding* for x. A new term tree $f' = f\{x := [g, \sigma]\}$ is obtained by applying the binding $x := [g, \sigma]$ to f in the following way. For the variable $h = [v_0, v_1, \ldots, v_\ell]$, we attach g to f by removing the variable h from H_f and by identifying the vertices $v_0, v_1, \ldots, v_\ell$ with the vertices $u_0, u_1, \ldots, u_\ell$ of g in this order. We define a new ordering $<_v^{f'}$ on every vertex v in f' in the following natural way. Suppose that v has more than one child and let v' and v'' be two children of v in f'. We note that $v_i = u_i$ for any $0 \leq i \leq \ell$. (1) If $v, v', v'' \in V_g$ and $v' <_v^g v''$, then $v' <_v^{f'} v''$. (2) If $v, v', v'' \in V_f$ and $v' <_v^f v''$, then $v' <_v^{f'} v''$. (3) If $v = v_0 (= u_0)$, $v' \in V_f - \{v_1, \ldots, v_\ell\}$, $v'' \in V_g$, and $v' <_v^f v_1$, then $v' <_v^{f'} v''$. (4) If $v = v_0 (= u_0)$, $v' \in V_f - \{v_1, \ldots, v_\ell\}$, $v'' \in V_g$, and $v_\ell <_v^f v'$, then $v'' <_v^{f'} v'$. In Fig. 2, we give an example of the new ordering on vertices in a term tree.

A *substitution* θ is a finite collection of bindings $\{x_1 := [g_1, \sigma_1], \cdots, x_n := [g_n, \sigma_n]\}$, where x_i's are mutually distinct variable labels in X. The term tree $f\theta$, called the *instance* of f by θ, is obtained by applying all the bindings $x_i := [g_i, \sigma_i]$ on f. Lastly we define the root of the resulting term tree $f\theta$ as the root of f. Consider the examples in Fig. 1. Let $\theta = \{x_1 := [g_1, [u_1, w_1]], x_2 := [g_2, [u_2, w_2]], x_3 := [g_3, [u_3, w_3, w_3']]\}$ be a substitution, where g_1, g_2, and g_3 are trees. Then the instance $t'\theta$ of the term tree t' by θ is the tree T_1.

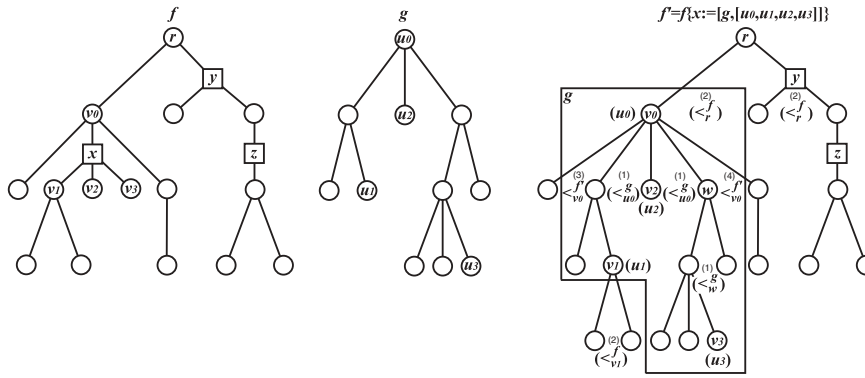


Fig. 2. The new ordering on vertices in the term tree $f' = f\{x := [g, [u_0, u_1, u_2, u_3]]\}$.

Definition 2. Let Λ be a set of edge labels. The *term tree language* $L_\Lambda(t)$ of a term tree $t \in \mathcal{OTT}^{*,*}_\Lambda$ is defined as $\{s \in \mathcal{OT}_\Lambda \mid s \equiv t\theta \text{ for a substitution } \theta\}$. The class $\mathcal{OTTL}^{*,*}_\Lambda$ of term tree languages is defined as $\{L_\Lambda(t) \mid t \in \mathcal{OTT}^{*,*}_\Lambda\}$.

3 A Polynomial Time Algorithm for Solving Membership Problem for $\mathcal{OTT}^{*,*}_\Lambda$

In this section, we give a polynomial time algorithm MATCHING (Fig.8) for Membership Problem for $\mathcal{OTT}^{*,*}_\Lambda$ where Λ is any finite or infinite set of edge labels.

Membership Problem for $\mathcal{OTT}^{*,*}_\Lambda$
Instance: A tree $T \in \mathcal{OT}_\Lambda$ and a term tree $t \in \mathcal{OTT}^{*,*}_\Lambda$.
Question: Is T included in $L_\Lambda(t)$?

3.1 C-Sets

Let $T = (V_T, E_T)$ and $t = (V_t, E_t, H_t)$ be a tree and a term tree, respectively. A term tree t is said to *match* a tree T if T is included in $L_\Lambda(t)$. First, we search all vertices of a term tree t by the breadth-first search method and give all the vertices the sequential numbers from 1 to $|V_t|$ in the search order. We call these numbers *vertex identifiers* (VIDs for short). Then the VID of the root of t is 1 and for any variable h the children of h have consecutive VIDs . Below we identify a vertex with the VID of it. For two integers i and j $(i \leq j)$, we denote by $[i, j]$ the interval consisting of all integers between i and j. For a variable h of t, an interval of VIDs is said to be a *P-interval* if it represents all VIDs of the child ports of h. A *P-subinterval* is a subinterval of a P-interval. For two P-subintervals $[i, j]$ and $[i', j']$, we say that $[i', j']$ is a *P-subinterval* of $[i, j]$ if $i \leq i' \leq j' \leq j$.

Definition 3. Let S be a set of VIDs and P-subintervals. Then S is said to be a *correspondence-set* (C-set for short) if for any two P-subintervals L, L' $(L \neq L')$ in S, L is not a P-subinterval of L' and L' is not a P-subinterval of L.

Example 1. We consider a vertex with five children whose VIDs are $1, 2, 3, 4, 5$. Then $\{[1,2], 3, [4,4]\}$, $\{[2,4], 2, 5\}$, and $\{[2,3], 3, [3,4], 5\}$ are C-sets.

The intended meaning of a C-set is stated in the proof of Theorem 1. C-sets are used in the procedures MATCHING and C_SET_ATTACHING (Fig. 8).

3.2 A Data Structure for C-Sets

Let $t = (V_t, E_t, H_t)$ be a term tree and CS a C-set. We define the following arrays:

$$last(i) = \begin{cases} \max\{j \mid [i,j] \text{ is a P-subinterval of some P-subinterval in } CS\} \\ \qquad \text{if } CS \text{ contains a P-subinterval containing } i, \\ 0 \qquad \text{otherwise;} \end{cases}$$

$$isvid(i) = \begin{cases} i & \text{if } CS \text{ contains } i \text{ as a VID,} \\ 0 & \text{otherwise.} \end{cases}$$

For vertices with VIDs in Example 1, CS_1, CS_2, CS_3 are represented as follows.

$CS_1 = \{[1,2], 3, [4,4]\}$ $\quad$ $CS_2 = \{[2,4], 2, 5\}$ $\quad$ $CS_3 = \{[2,3], 3, [3,4], 5\}$

VID	1	2	3	4	5
last	2	2	0	4	0
isvid	0	0	3	0	0

VID	1	2	3	4	5
last	0	4	4	4	0
isvid	0	2	0	0	5

VID	1	2	3	4	5
last	0	3	4	4	0
isvid	0	0	3	0	5

We can reconstruct a C-set from the data structure of it because any P-subinterval in the C-set is not a P-subinterval of another P-subinterval in the C-set.

3.3 Subproblems

Definition 4. Let $CS_1, \ldots, CS_m$ be m C-sets. A *C-set-sequence* is a sequence of C-sets $(CS_1, \ldots, CS_m)$. Let $[i_h, j_h]$ be a P-interval of a variable h. We say that $(CS_1, \ldots, CS_m)$ *covers* $[i_h, j_h]$ if there are two index sequences $1 \leq p_1 < p_2 < \cdots < p_{m'} \leq m$ and $i_h = q_1 < q_2 < \cdots < q_{m'} \leq q_{m'+1} - 1 = j_h$ for some $m' \geq 1$ such that $q_i \in CS_{p_i}$ with $q_i + 1 = q_{i+1}$ or $[q_i, q_{i+1} - 1]$ is a P-subinterval of some P-subinterval in CS_{p_i} for all i $(1 \leq i \leq m')$.

Example 2. For vertices in Example 1, a sequence of C-sets $(CS_1, CS_2, CS_3) = (\{[1,2], 3, [4,4]\}, \{[2,4], 2, 5\}, \{[2,3], 3, [3,4], 5\})$ covers a P-interval $[1,5]$ for $m = 3, m' = 3, p_1 = 1, p_2 = 2, p_3 = 3, q_1 = 1, q_2 = 3, q_3 = 5$.

Procedure BASIC_COVER($(CS_1, \ldots, CS_m)$: C-set-sequence, $[i_h, j_h]$: P-interval);
begin
  $i := i_h$;
  for $k := 1$ **to** m **do begin**
    $i := \max\{last_k(i) + 1, isvid_k(i) + 1, i\}$;
    if $i = j_h + 1$ **then** return k
  end;
  return 0
end;

Fig. 3. Procedure Basic_Cover: $last_j(i)$ and $isvid_j(i)$ represent $last(i)$ and $isvid(i)$ entries of CS_j, respectively.

First we consider the following several problems:

Basic_Cover
Input: a C-set-sequence $(CS_1, \ldots, CS_m)$ and a P-interval $[i_h, j_h]$.
Output: Return the smallest index k $(1 \leq k \leq m)$ such that $(CS_1, \ldots, CS_k)$ covers $[i_h, j_h]$, if there is not such an index, return 0.

This problem is computable in $O(m)$ time by the following straightforward procedure described in Fig. 3.

Multiple_Cover
Input: a C-set-sequence $(CS_1, \ldots, CS_m)$ and m' P-intervals $[i_1, j_1], \ldots, [i_{m'}, j_{m'}]$, where $j_\ell + 1 = i_{\ell+1}$ for all ℓ $(1 \leq \ell \leq m' - 1)$.
Output: Return the smallest index k $(1 \leq k \leq m)$ such that $(CS_1, \ldots, CS_k)$ covers $[i_1, j_1] \cup \cdots \cup [i_{m'}, j_{m'}]$, if there is not such an index, return 0.

This problem is computable in $O(m)$ time straightforwardly by using Procedure MULTIPLE_COVER (Fig. 4). We use the following notations to describe our algorithms. For a sequence of elements $\mathcal{A} = (A_1, A_2, \ldots, A_m)$ and $n, n' \geq 0$,

$$\mathcal{A}^{[n]} = \begin{cases} (A_1, A_2, \ldots, A_n) & \text{if } n \leq m, \\ () & \text{if } n = 0, \\ \mathcal{A} & \text{otherwise;} \end{cases}$$

$$\mathcal{A}_{[n']} = \begin{cases} (A_{m-n'+1}, \ldots, A_{m-1}, A_m) & \text{if } n' \leq m, \\ () & \text{if } n' = 0, \\ \mathcal{A} & \text{otherwise;} \end{cases}$$

$$\mathcal{A}[n, n'] = \begin{cases} (A_{n+1}, \ldots, A_{m-n'}) & \text{if } n + 1 \leq m - n', \\ () & \text{otherwise.} \end{cases}$$

Definition 5. Let $(CS_1, \ldots, CS_m)$ be a C-set-sequence and $[i_h, j_h]$ a P-interval of a variable h. We say that $(CS_1, \ldots, CS_m)$ *strictly covers* $[i_h, j_h]$ if there are two indices k, ℓ with $1 \leq k \leq \ell \leq m$ and $\ell - k = j_h - i_h$ such that for each i $(i_h \leq i \leq j_h)$ CS_{k+i-i_h} includes a VID i.

```
Procedure MULTIPLE_COVER((CS_1, ..., CS_m): C-set-sequence,
    [i_1, j_1], ..., [i_m', j_m']: P-intervals);
begin
  CS := (CS_1, ..., CS_m); k := 0;
  for ℓ := 1 to m' do begin
    k' :=BASIC_COVER(CS,[i_ℓ, j_ℓ]);
    if k' = 0 then return 0 else begin CS := CS[k', 0]; k := k + k' end
  end;
  return k
end;
```

Fig. 4. Procedure Multiple_Cover.

```
Procedure STRICT_COVER((CS_1, ..., CS_m): C-set-sequence,
   [i_h, j_h]: interval of VIDs );
begin
  for s := 1 to m − j_h + i_h do
    for k := s to s + j_h − i_h do begin
      if isvid_k(i_h + k − s) = 0 then break;
      if k = s + j_h − i_h then return k
    end;
  return 0
end;
```

Fig. 5. Procedure Strict_Cover.

Strict_Cover
Input: a C-set-sequence $(CS_1, \ldots, CS_m)$ and an interval $[i_h, j_h]$ where i_h and j_h are VIDs $(i_h \leq j_h)$.
Output: Return the smallest index ℓ $(j_h - i_h < \ell \leq m)$ such that $(CS_1, \ldots, CS_\ell)$ strictly covers $[i_h, j_h]$, if there is not such an index, return 0.

This problem is computable in $O(m(j_h - i_h + 1))$ time by the procedure described in Fig. 5.

All_Maximal_Subinterval
Input: a C-set-sequence $(CS_1, \ldots, CS_m)$ and a P-interval $[i_h, j_h]$.
Output: Return the C-set consisting of all P-subintervals of $[i_h, j_h]$ which are covered by $(CS_1, \ldots, CS_m)$.

Example 3. Let $CS_1 = \{[1, 2], [2, 3]\}$, $CS_2 = \{[1, 2], 5\}$, $CS_3 = \{3, 4\}$, and $CS_4 = \{[2, 4], 5, [6, 6]\}$ be input C-sets. Let $[1, 6]$ be an input P-interval. Then the output C-set is $\{[1, 4], [2, 5], [5, 6]\}$.

This problem is computable in $O(m(j_h - i_h + 1))$ time by the procedure in Fig. 6.

```
Procedure ALL_MAXIMAL_SUBINTERVAL ((CS_1, ..., CS_m): C-set-sequence,
          [i_h, j_h]: P-interval);
begin
  for i := i_h to j_h do begin last(i) := max{last_1(i), isvid_1(i)}; isvid(i) := 0 end;
  for j := 2 to m do
    for i := i_h to j_h do
      v := last(i);
      if v = 0 then
        last(i) := min{j_h, max{last_j(i), isvid_j(i)}}
      else if j_h ≤ v then
        last(i) := j_h
      else
        last(i) := min{j_h, max{last_j(v + 1), isvid_j(v + 1), v}};
  return last and isvid for a new C-set CS
end;
```

Fig. 6. Procedure All_Maximal_Subinterval: Let CS be the output C-set and $last(i)$ and $isvid(i)$ show the entries of $last$ and $isvid$ of VID i of CS.

3.4 C-Set-Attaching Rules

In this section, we define a *C-set-attaching rule* which plays an important role in our algorithm. The VID of a vertex u' is denoted by $I(u')$.

Definition 6. Let u' be a vertex of a term tree t and $u'_1, \ldots, u'_{m'}$ all children of u'. We assume that u'_i is smaller than u'_j for any $i < j$ in the order of the children of u. Let $f_1, \ldots, f_{m''}$ be all edges and variables which include u' and some of $u'_1, \ldots, u'_{m'}$. We assume that for any $1 \leq i', j' \leq m''$, if there are two indices i, j $(1 \leq i < j \leq m')$ such that $u'_i \in f_{i'}$ and $u'_j \in f_{j'}$ then $i' \leq j'$. The *C-set-attaching rule* of u' is $I(u') \leftarrow J(f_1), \ldots, J(f_{m''})$ where for $i = 1, \ldots, m''$,

$$J(f_i) = \begin{cases} I(u'_k) & \text{if } f_i = \{u', u'_k\} \in E_t \\ & \text{for some } k \ (1 \leq k \leq m'), \\ [I(u'_k), I(u'_{k+\ell})] & \text{if } f_i = [u', u'_k, u'_{k+1} \ldots, u'_{k+\ell}] \in H_t \\ & \text{for some } k, \ell \ (1 \leq k \leq k + \ell \leq m'). \end{cases}$$

3.5 How to Use C-Set-Attaching Rules

We explain how to apply $I(u') \leftarrow J(f_1), \ldots, J(f_{m''})$ to a vertex v with m children of a tree T. Let $v_1, \ldots, v_m$ be all the children of v, and $CS(v_1), \ldots, CS(v_m)$ all C-sets of $v_1, \ldots, v_m$, respectively.

Then we consider the following problem:

Rule_Matching
Input: C-set-sequence $(CS_1, \ldots, CS_m)$ and a sequence of VIDs and P-intervals $(J_1, \ldots, J_{m''})$.
Question: Is there an index subsequence $1 = \ell_1 < \ell_2 < \cdots < \ell_{m''} \leq \ell_{m''+1} - 1 = m$ satisfying the following conditions for each $1 \leq i \leq m'$?

Procedure RULE_MATCHING($\mathcal{C} = (CS_1, \ldots, CS_m)$, $\mathcal{J} = (J_1, \ldots, J_{m''})$);
begin
if all members in $\mathcal{J}$ are VIDs **then**
if $m = m''$ and STRICT_COVER($\mathcal{C}, [J_1, , J_m]) = m$ **then** return "*yes*"
else return "*no*";
$n := \max\{k \mid$ all members in $\mathcal{J}^{[k]}$ are VIDs$\}$;
if $n > 0$ and STRICT_COVER($\mathcal{C}^{[n]}, [J_1, J_n]) = 0$ **then** return "*no*";
$n' := \max\{k' \mid$ all members in $\mathcal{J}_{[k']}$ are VIDs$\}$;
if $n' > 0$ and STRICT_COVER($\mathcal{C}_{[n']}, [J_{m-n'+1}, J_m]) = 0$ **then** return "*no*";
return RULE_MATCHING_SUB($\mathcal{C}[n, n'], \mathcal{J}[n, n']$)
end;

Procedure RULE_MATCHING_SUB($\mathcal{C} = (CS_1, \ldots, CS_m)$, $\mathcal{J} = (J_1, \ldots, J_{m''})$);
begin
if all members in $\mathcal{J}$ are P-intervals **then**
if MULTIPLE_COVER($\mathcal{C}, \mathcal{J}) \neq 0$ **then** return "*yes*" **else** return "*no*";
$n := \max\{k \mid$ all members in $\mathcal{J}^{[k]}$ are P-intervals$\}$;
$r :=$MULTIPLE_COVER($\mathcal{C}, \mathcal{J}^{[n]}$);
if $r = 0$ **then** return "*no*";
$n' := \max\{k' \mid$ all members in $\mathcal{J}[n, 0]^{[k']}$ are VIDs$\}$;
$r' :=$STRICT_COVER($\mathcal{C}[r, 0], \mathcal{J}[n, 0]^{[n']}$);
if $r' = 0$ **then** return "*no*";
return RULE_MATCHING_SUB($\mathcal{C}[r + r', 0], \mathcal{J}[n + n', 0]$)
end;

Fig. 7. Procedure RULE_MATCHING.

(*1*) if J_i is a VID then $J_i \in CS_{\ell_i}$ and $\ell_i + 1 = \ell_{i+1}$, and
(*2*) if J_i is a P-interval then $CS_{\ell_i}, \ldots, CS_{\ell_{i+1}-1}$ covers J_i.

Procedure RULE_MATCHING described in Fig. 7 computes the above problem in $O(mm'')$ time.

3.6 A Matching Algorithm

We repeatedly attach a C-set to vertices of a tree T from the leaves to the root by using C-set-attaching rules. The entire algorithm MATCHING is described in Fig. 8.

We show that the C-set attached to the root of a tree T contains the VID of the root of a term tree t if and only if t matches T. In Fig. 9, when the term tree t and the tree T are given, $Rule(t)$ denotes the set of all C-set-attaching rules of t. After Procedure MATCHING terminates, we can get the resulting C-sets of T.

Procedure MATCHING (Fig. 8) is for the case of $|\Lambda| = 1$. We can easily extend the algorithm for the case of $|\Lambda| \geq 2$ by checking edge labels of a term tree and a tree in applying C-set-attaching rules.

Theorem 1. *Membership Problem for $\mathcal{OTT}^{*,*}_{\Lambda}$ is computable in polynomial time for any set of edge labels Λ.*

Procedure MATCHING(term tree t in $\mathcal{OTT}_{\Lambda}^{*,*}$ with root r, tree T in $\mathcal{OT}_{\Lambda}$ with root R);
begin
  Let $Rule(t)$ be the set of all C-set-attaching rules of t;
  For each leaf ℓ of T, attach the C-set $\{I(\ell') \mid \ell'$ is a leaf of $t\}$ to ℓ;
  while there exists a vertex v of T
      such that v is not attached with any C-set and
      all children of v are attached with C-sets
    do C_SET_ATTACHING$(v, Rule(t))$;
  if the C-set attached to R includes the vertex identifier $I(r)$ of r
  then report that t matches T
  else report that t does not match T
end.

Procedure C_SET_ATTACHING(vertex v of T, set of C-set-attaching rules $Rule$);
begin
  $C := \emptyset$;
  Let $v_1, \ldots, v_m$ be all ordered children of v in T;
  Let $CS(v_1), \ldots, CS(v_m)$ be the C-sets attached to $v_1, \ldots, v_m$, respectively;
  foreach $I(u') \leftarrow J(f_1), \cdots, J(f_{m''})$ in $Rule$ **do begin**
    if RULE_MATCHING$((CS(v_1), \ldots, CS(v_m)), (J(f_1), \cdots, J(f_{m''}))) =$"$yes$" **then**
      $C := C \cup \{I(u')\}$;
    for $i := 1$ **to** m'' **do**
      if $J(f_i)$ is a P-interval **then**
        $C := C \cup$ ALL_MAXIMAL_SUBINTERVAL$((CS(v_1), \ldots, CS(v_m)), J(f_i))$
  end;
  Attach C to v as its C-set
end.

Fig. 8. Procedure MATCHING: an algorithm for the membership problem for the set $\mathcal{OTT}_{\Lambda}^{*,*}$ of term trees.

Proof. Let $CS(v)$ be the C-set attached to a vertex v of a tree T after the procedure C_SET_ATTACHING for v terminates. For a vertex u of a term tree t , we denote by $t[u]$ the term tree consisting of u and all descendants of u. Similarly for a vertex v of a tree T, we denote by $T[v]$ the tree consisting of v and all descendants of v. By induction on applications of C-set-attaching rules, we can show that the following claims.

Claim 1. $I(u') \in CS(v)$ if and only if $t[u']$ matches the subtree $T[v]$ of T.

Claim 2. $[I(u'), I(u')] \in CS(v)$ if and only if (i) $t[u']$ matches $T[v]$ or (ii) $t[u']$ matches $T[v']$ for a descendant v' of v.

Claim 3. $[I(u'), I(u'')] \in CS(v)$ $(u' \neq u'')$ if and only if there exist distinct $I(u'') - I(u') + 1$ descendants v_i $(I(u') \leq i \leq I(u''))$ of v such that (i) v_i is not descendant of v_j $(I(u') \leq i, j \leq I(u''))$ and (ii) $t[u_i]$ matches $T[v_i]$ for all i $(I(u') \leq i \leq I(u''))$ where u_i is a vertex having a VID i.

Thus we show that the C-set attached to the root of a tree T contains the VID of the root of a term tree t if and only if t matches T.

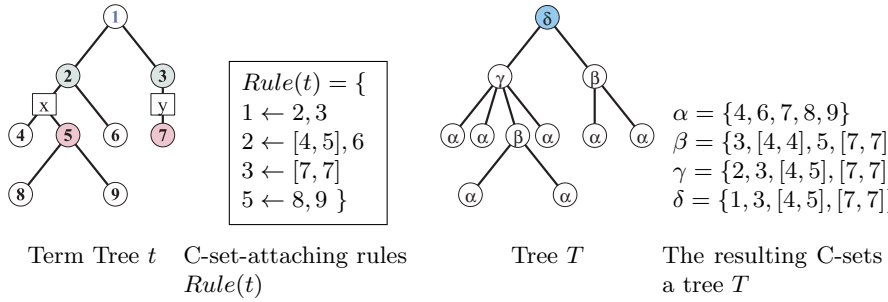


Fig. 9. The C-set-attaching rules of a term tree t. The resulting C-sets of a tree T after the procedure MATCHING terminates.

Let n and N be the numbers of vertices of a term tree t and a tree T, respectively. Let v be an internal vertex of T and $d(v)$ the number of children of v. As stated in section 3.5, $O(d(v)m'')$ time is needed for one application of a C-set-attaching rule $I(u') \leftarrow J(f_1), \ldots, J(f_{m''})$. And Procedure ALL_MAXIMAL_SUBINTERVAL needs the same total time. Then in order to make the C-set $CS(u)$, we need

$$O(d(v) \sum_{I(u') \leftarrow J(f_1),\ldots,J(f_{m''}) \in Rule(t)} m'') = O(d(v)n).$$

Thus the total time complexity of Procedure MATCHING is $O(n \sum_{v \in V_T} d(v)) = O(nN)$. □

4 Implementation and Experimental Results

In order to show the efficiency of our matching algorithm in Fig. 8 for a collection of real Web pages, we have implemented it in Java on a PC (OS: WindowsXP, CPU: 1.7GHz Xeon).

We report experimental results on our matching algorithm in Fig. 10. We constructed the set of input trees as follows. We gave keywords "java" and "coffee" to the search engine *google*(http://www.google.com) and made the list of top 480 active URLs which were returned by the engine. We converted an HTML file of a URL in the list to a tree structured data of some specified HTML tags. The number of vertices of an input tree is displayed in the x-axis in Exp.1 and 2. The average execution time (run time, millisecond) of our matching algorithm for a term tree and a tree is displayed in the y-axis. As input term trees, we gave term trees consisting of only variables, i.e. term trees with no edge, in order to evaluate the effect of the numbers of vertices of term trees and trees on the performance of the algorithm.

In Exp.1, we have experiments on all 4 term trees of 4 vertices and a tree and report the average execution time of our matching algorithm. For example, a point in the figure shows that the average execution time of term trees of 4 vertices and a tree of 500 vertices is about 100 millisecond. In Exp.2, we

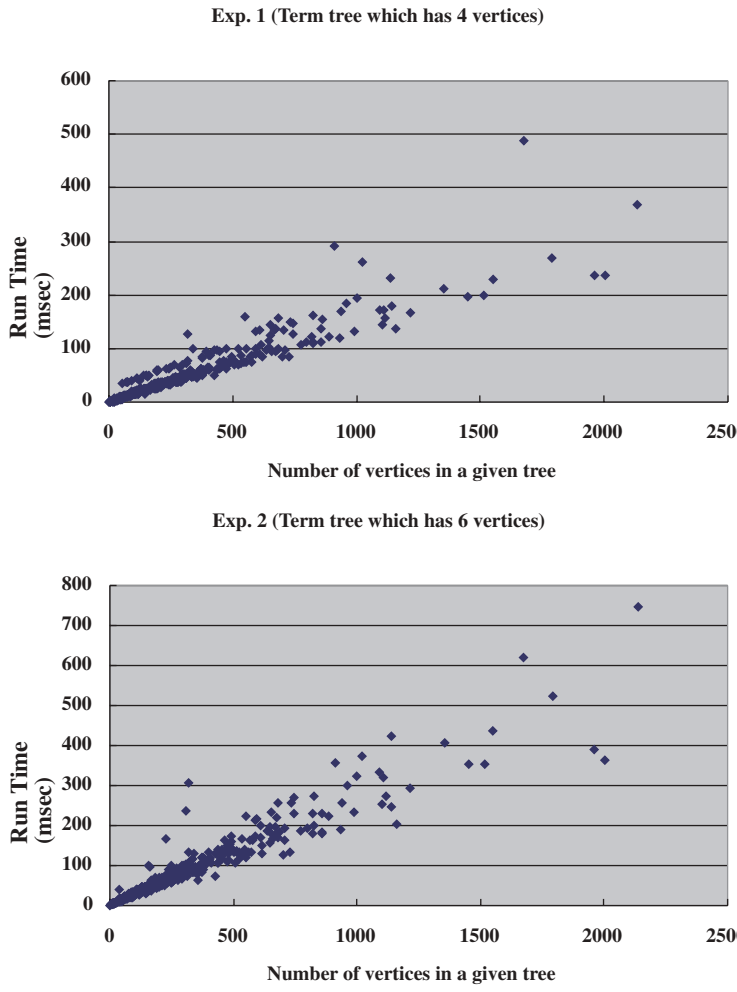


Fig. 10. Experimental results of our matching algorithm.

have experiments on all 42 term trees of 6 vertices and a tree and report the average execution time of our matching algorithm. These figures show that the execution time increases linearly in the number of vertices of trees and our matching algorithm is efficient.

5 Conclusions

We have studied data mining of structured ordered tree patterns from semistructured data. We have proposed a term tree as a representation of an ordered tree structured pattern in tree structured data or semistructured data. Toward design of efficient data mining tools for semistructured data, we have given an efficient matching algorithm for a term tree and a tree. Also we have reported some experimental results of the algorithm on HTML files of Web pages.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann, 2000.
2. T. R. Amoth, P. Cull, and P. Tadepalli. Exact learning of tree patterns from queries and counterexamples. *Proc. COLT-98, ACM Press*, pages 175–186, 1998.
3. T. R. Amoth, P. Cull, and P. Tadepalli. Exact learning of unordered tree patterns from queries. *Proc. COLT-99, ACM Press*, pages 323–332, 1999.
4. D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Science*, 21:46–62, 1980.
5. H. Arimura, H. Sakamoto, and S. Arikawa. Efficient learning of semi-structured data from queries. *Proc. ALT-2001, Springer-Verlag, LNAI 2225*, pages 315–331, 2001.
6. M. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. *Proceedings of the 14th Internatinal Conference on Data Engineering (ICDE-98), IEEE Computer Society*, pages 14–23, 1998.
7. S. Matsumoto, Y. Hayashi, and T. Shoudai. Polynomial time inductive inference of regular term tree languages from positive data. *Proc. ALT-97, Springer-Verlag, LNAI 1316*, pages 212–227, 1997.
8. S. Matsumoto, T. Shoudai, T. Miyahara, and T. Uchida. Learning of Finite Unions of Tree Patterns with Internal Structured Variables from Queries. *Proc. AI-2002, Springer-Verlag, LNAI (to appear)*, 2002.
9. T. Miyahara, T. Shoudai, T. Uchida, K. Takahashi, and H. Ueda. Polynomial time matching algorithms for tree-like structured patterns in knowledge discovery. *Proc. PAKDD-2000, Springer-Verlag, LNAI 1805*, pages 5–16, 2000.
10. T. Miyahara, T. Shoudai, T. Uchida, K. Takahashi, and H. Ueda. Discovery of frequent tree structuted patterns in semistructured web documents. *Proc. PAKDD-2001, Springer-Verlag, LNAI 2035*, pages 47–52, 2001.
11. T. Miyahara, Y. Suzuki, T. Shoudai, T. Uchida, K. Takahashi, and H. Ueda. Discovery of frequent tag tree patterns in semistructured web documents. *Proc. PAKDD-2002, Springer-Verlag, LNAI 2336*, pages 341–355, 2002.
12. T. Shinohara. Polynomial time inference of extended regular pattern languages. In *Springer-Verlag, LNCS 147*, pages 115–127, 1982.
13. T. Shoudai, T. Miyahara, T. Uchida, and S. Matsumoto. Inductive inference of regular term tree languages and its application to knowledge discovery. *Information Modelling and Knowledge Bases XI, IOS Press*, pages 85–102, 2000.
14. T. Shoudai, T. Uchida, and T. Miyahara. Polynomial time algorithms for finding unordered term tree patterns with internal variables. *Proc. FCT-2001, Springer-Verlag, LNCS 2138*, pages 335–346, 2001.
15. Y. Suzuki, R. Akanuma, T. Shoudai, T. Miyahara, and T. Uchida. Polynomial time inductive inference of ordered tree patterns with internal structured variables from positive data. *Proc. COLT-2002, Springer-Verlag, LNAI 2375*, pages 169–184, 2002.
16. Y. Suzuki, T. Shoudai, T. Miyahara, and T. Uchida. Ordered Term Tree Languages Which Are Polynomial Time Inductively Inferable from Positive Data. *Proc. ALT-2002, Springer-Verlag, LNAI (to appear)*, 2002.
17. K. Wang and H. Liu. Discovering structural association of semistructured data. *IEEE Trans. Knowledge and Data Engineering*, 12:353–371, 2000.

A Genetic Algorithms Approach to ILP

Alireza Tamaddoni-Nezhad and Stephen Muggleton

Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, UK
{atn,shm}@doc.ic.ac.uk

Abstract. In a previous paper we introduced a framework for combining Genetic Algorithms with ILP which included a novel representation for clauses and relevant operators. In this paper we complete the proposed framework by introducing a fast evaluation mechanism. In this evaluation mechanism individuals can be evaluated at genotype level (i.e. bit-strings) without mapping them into corresponding clauses. This is intended to replace the complex task of evaluating clauses (which usually needs repeated theorem proving) with simple bitwise operations. In this paper we also provide an experimental evaluation of the proposed framework. The results suggest that this framework could lead to significantly increased efficiency in problems involving complex target theories.

1 Introduction

Current ILP systems mostly employ deterministic search methods to examine the refinement space of clauses and use different kind of syntactic biases and heuristics (e.g. greedy methods) to cope with the complexity of the search, which otherwise is intractable. Using these biases usually limits the exploration power of the search and may lead to local optima. On the other hand, more powerful search methods are required for dealing with large search spaces (for example in multi-clause learning). Genetic Algorithms (GAs) have great potential for this purpose and it is likely that a combination of ILP and GAs can overcome the limitation of each individual method and can cope with some of the complexities of real-world applications.

In [26] a framework for combining Genetic Algorithms with ILP was introduced and a novel binary representation and relevant operators were discussed. It was shown that the proposed representation has interesting properties in terms of first-order concept learning. For example, it was shown that essential operations on clauses, such as unification and anti-unification [22,21], can be achieved by simple bitwise operations (e.g. and/or) on binary strings.

In this paper we complete the proposed framework by introducing a fast evaluation mechanism for evaluating individuals at genotype level. In this paper we also explain an implementation of the proposed framework together with an empirical evaluation of the implemented system.

The paper is organised as follows. Section 2 reviews the proposed framework and also provides the definition and theorems which are needed in the next

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 285–300, 2003.

sections. Section 3 introduces a new evaluation mechanism for clauses. In this section we show that evaluating a clause can be done by simple and fast operations. Section 4 introduces stochastic refinement for directing genetic operators. Implementation and evaluation are explained in section 5. Related work and similar systems are discussed in section 6. Finally, section 7 concludes this paper and gives some directions for further research.

2 Representation, Encoding and Operators

In this section, we review the proposed binary encoding for first-order clauses and some of its properties. We also provide a summary of definitions and theorems which will be required in the rest of this paper. Proofs and more details about the representation and operators can be found in [26].

Figure 1.a shows the main idea of the binary representation by a simple example. Consider a clause with n variable occurrences. The relationships between these n variable occurrences can be represented by a graph having n vertices in which there exists an edge between vertices v_i and v_j if ith and jth variable occurrences in the clause represent the same variable. For example variable bindings in clause *p(X,Y):-q(X,Z),r(Z,Y)* represent an undirected graph and this clause can be represented by a binary matrix as shown in Figure 1.a. In this matrix entry m_{ij} is 1 if ith and jth variable occurrences in the clause represent the same variable and m_{ij} is 0 otherwise. This representation has interesting properties which can be exploited by a genetic algorithm for searching the refinement space of a clause.

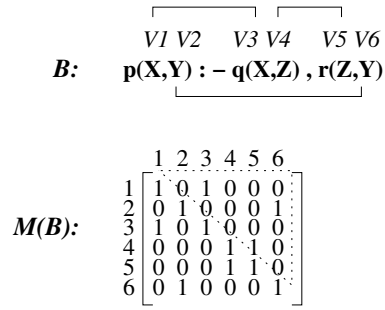
Definition 1 (Binding Set). *Let B and C both be clauses. C is in binding set $\mathcal{B}(B)$ if there exists a variable substitution*[1] *θ such that $C\theta = B$.*

Definition 2 (Binding Matrix). *Suppose B and C are both clauses and there exists a variable substitution θ such that $C\theta = B$. Let C have n variable occurrences representing variables $\langle v_1, v_2, \ldots, v_n \rangle$. The binding matrix of C is an $n \times n$ matrix M in which m_{ij} is 1 if there exist variables v_i, v_j and u such that v_i/u and v_j/u are in θ and m_{ij} is 0 otherwise. We write $M(v_i, v_j) = 1$ if $m_{ij} = 1$ and $M(v_i, v_j) = 0$ if $m_{ij} = 0$.*

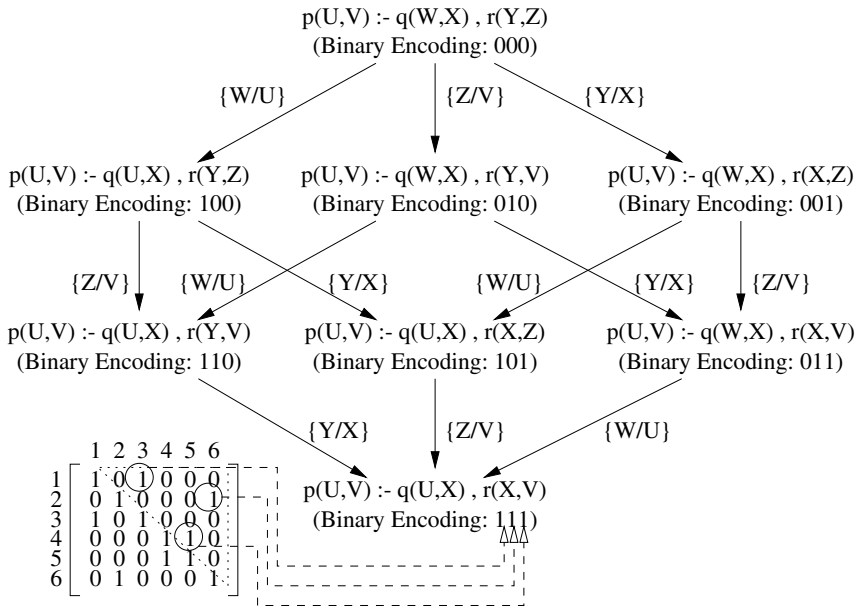
Definition 3 (Normalized Binding Matrix). *Let M be an $n \times n$ binary matrix. M is in the set of normalized binding matrices $\mathcal{M}_n$ if M is symmetric and for each $1 \leq i \leq n$, $1 \leq j \leq n$ and $1 \leq k \leq n$, $m_{ij} = 1$ if $m_{ik} = 1$ and $m_{kj} = 1$.*

Definition 4 (Mapping Function M(C)). *The mapping function $M : \mathcal{B}(B) \rightarrow \mathcal{M}_n$ is defined as follows. Given clause $C \in \mathcal{B}(B)$ with n variable occurrences representing variables $\langle v_1, v_2, \ldots, v_n \rangle$, $M(C)$ is an $n \times n$ binary matrix in which m_{ij} is 1 if variables v_i and v_j are identical and m_{ij} is 0 otherwise.*

[1] Substitution $\theta = \{v_i/u_j\}$ is a variable substitution if all v_i and u_j are variables.



(a) Binding matrix for clause: *p(X, Y):-q(X,Z),r(Z, Y)*



(b) A substitution lattice bounded below by clause *p(X,Y):-q(X,Z),r(Z,Y)* and binary encoding for each clause

Fig. 1. A binary representation for clauses and the relationship between the binary strings and the substitution ordering between clauses.

Definition 5 (Mapping Function C(M)). *The mapping function* $C : \mathcal{M}_n \rightarrow \mathcal{B}(B)$ *is defined as follows. Given a normalized* $n \times n$ *binding matrix* M, $C(M)$ *is a clause in* $\mathcal{B}(B)$ *with* n *variable occurrences* $\langle v_1, v_2, \ldots, v_n \rangle$, *in which variables* v_i *and* v_j *are identical if* m_{ij} *is 1.*

Definition 6 (Matrix Subset). *Let P and Q be in $\mathcal{M}_n$. It is said that $P \subseteq Q$ if for each entry $p_{ij} \in P$ and $q_{ij} \in Q$, p_{ij} is 1 if q_{ij} is 1. $P = Q$ if $P \subseteq Q$ and $Q \subseteq P$. $P \subset Q$ if $P \subseteq Q$ and $P \neq Q$.*

Definition 7 (Subsumption and Substitution). *Clause C subsumes clause D, $C \succeq D$ if there exists a (variable) substitution θ such that $C\theta \subseteq D$ (i.e. every literal in $C\theta$ is also in D). C properly subsumes D, $C \succ D$ if $C \succeq D$ and $D \not\succeq C$. Clause D is a substitution of clause C, $C \sqsupseteq D$ if there exists a (variable) substitution θ' such that $C\theta' = D$ (i.e. every literal in $C\theta'$ corresponds to a literal in D). D is a proper instance of C, $C \sqsupset D$ if $C \sqsupseteq D$ and $D \not\sqsupseteq C$.*

Because of the substitution order between clauses (which is a quasi-order) the search space (or refinement space) can be modelled as a sub-lattice of subsumption [21]. The following theorem represents the relationship between binary matrices and the substitution order of clauses. However, because these theorems also hold for the subsumption order we use the subsumption notation which is more general.

Theorem 1. *For each clause B and matrices M_1 and M_2 in $\mathcal{M}_n$ such that $C(M_1) \in \mathcal{B}(B)$ and $C(M_2) \in \mathcal{B}(B)$, $C(M_1) \succ C(M_2)$ if $M_1 \subset M_2$.*

Example 1. Figure 1.b shows a substitution lattice bounded below by the clause p(X,Y):-q(X,Z),r(Z,Y). This lattice shows the substitution ordering between clauses which are in the same binding set. According to Theorem 1, binding matrix of each clause which subsumes the bottom clause must be a subset of the binding matrix of the bottom clause. Hence, each clause in this search space can be encoded by 3 bits.

The proposed binary representation can be used to encode a substitution lattice (bounded below by a bottom-clause) in a compact way. Moreover, this encoding reduces the redundancy which we usually have in a first-order representation. However, we may still have redundant binary representation for clauses with more than one literal per predicate. To avoid this redundancy, one solution is to re-order literals with the same predicate symbol in such a way that the corresponding binary number is minimal. Using this strategy ensures that the binary encoding for these clauses is not redundant. We assume that this condition is hold in the following definitions and theorems. Another property of the proposed representation is that mgi(most general instance) and lgg(least general generalization) which are also known as unification and anti-unification operations on clauses [22,21] can be achieved by simple bitwise operations on the binary encoding of clauses. In the following, first we introduce *mgi* and *lgg* operations for clauses.

Definition 8 (mgi and lgg). *Clauses E and F are respectively a common instance and a common generalization of clauses C and D if and only if $C, D \succeq E$ and $F \succeq C, D$. $mgi(C, D)$ and $lgg(C, D)$ are the most general instance and the*

least general generalization for clauses C and D if and only if for every common instance E and common generalization F it is the case that $mgi(C, D) \succeq E$ and $F \succeq lgg(C, D)$.

Example 2. In Figure 1.b clause p(U,V):-q(U,X),r(X,Z) is the *mgi* of clauses p(U,V):-q(U,X),r(Y,Z) and p(U,V):-q(W,X),r(X,Z), and p(U,V):-q(W,X),r(Y,V) is the *lgg* of clauses p(U,V):-q(U,X),r(Y,V) and p(U,V):-q(W,X),r(X,V).

Definition 9 (Matrix AND). *Let M_1 and M_2 be in $\mathcal{M}_n$. $M = (M_1 \wedge M_2)$ is an $n \times n$ matrix and for each $a_{ij} \in M$, $b_{ij} \in M_1$ and $c_{ij} \in M_2$, $a_{ij} = 1$ if $b_{ij} = 1$ and $c_{ij} = 1$ and $a_{ij} = 0$ otherwise.*

Similar to AND operator, OR operator $(M_1 \vee M_2)$ is constructed by bitwise OR-ing of M_1 and M_2 entries.

Definition 10 (Matrix OR). *Let M_1 and M_2 be in $\mathcal{M}_n$. $M = (M_1 \vee M_2)$ is an $n \times n$ matrix and for each $a_{ij} \in M$, $b_{ij} \in M_1$ and $c_{ij} \in M_2$, $a_{ij} = 1$ if $b_{ij} = 1$ or $c_{ij} = 1$ and $a_{ij} = 0$ otherwise.*

Theorem 2. *For each clause B and matrices M_1, M_2 and M in $\mathcal{M}_n$ such that $C(M_1) \in \mathcal{B}(B)$, $C(M_2) \in \mathcal{B}(B)$ and $C(M) \in \mathcal{B}(B)$, $C(M) = lgg(C(M_1), C(M_2))$ if $M = (M_1 \wedge M_2)$.*

Theorem 3. *For each clause B and matrices M_1, M_2 and M in $\mathcal{M}_n$ such that $C(M_1) \in \mathcal{B}(B)$, $C(M_2) \in \mathcal{B}(B)$ and $C(M) \in \mathcal{B}(B)$, $C(M) = mgi(C(M_1), C(M_2))$ if $M = M_1 \vee M_2$.*

Example 3. In Figure 1.b, *lgg* and *mgi* of any two clauses can be obtained by AND-ing and OR-ing of their binary strings.

In summary, the proposed binary representation can be used to encode a substitution lattice bounded below by a bottom clause. Essential operations on clauses can be achieved by bit-wise operation on binary strings. These operations can be considered as task-specific genetic operators which together with conventional genetic operators (i.e. mutation and crossover) can be used to search the refinement space of clauses. This issue will be discussed in section 4. In the next section we show how the properties of the proposed representation can be used to evaluate individuals (i.e. binary strings) at genotype level without mapping them into corresponding clauses.

3 Evaluation Mechanism

The usual way for evaluating a hypothesis in first-order concept learning systems is to repeatedly call a theorem prover (e.g. Prolog interpreter) on training examples to find out positive and negative coverage of the hypothesis. This step is known to be a complex and time-consuming task in first-order concept learning. In the case of genetic-based systems this situation is even worse, because we

need to evaluate a population of hypotheses in each generation. This problem is another important difficulty when applying GAs in first-order concept learning.

In this section we introduce a method which is intended to replace the complex task of evaluating clauses with bitwise operations on binary strings. This idea is similar to data-compilation method used by the attribute-based learning system GIL [9]. This system retains binary coverage vectors for all possible features (attributes and values) which can appear in a rule. This introduces a database which can be used for computing the coverage set of each rule by bitwise operations on the coverage vectors of participating features. However, in our case there is no need to maintain such a database. We show that by maintaining the coverage sets for a small number of clauses and by doing bitwise operations we can compute the coverage for other binary strings without mapping them into the corresponding clauses. This property is based on the implicit subsumption order which exists in the binary representation. In the following, first we define the cover-vector approach for representing coverage of a clause on training examples.

Definition 11 (Cover Sets and Cover Vectors). *Let C be a clause and $E^+ = \{e_1^+, e_2^+, \ldots, e_k^+\}$ and $E^- = \{e_1^-, e_2^-, \ldots, e_l^-\}$ be the set of positive and negative training examples respectively*[2]*. e_i^+ is in the positive cover set $\mathcal{P}(C)$ if $C \succeq e_i^+$. Similarly, e_j^- is in the negative cover set $\mathcal{N}(C)$ if $C \succeq e_j^-$. The positive cover vector $\mathcal{PV}(C)$ is a k-bit binary string in which bit i is 1 if $e_i^+ \in \mathcal{P}(C)$ and 0 otherwise. Similarly, the negative cover vector $\mathcal{NV}(C)$ is a l-bit binary string in which bit j is 1 if $e_j^- \in \mathcal{N}(C)$ and 0 otherwise.*

Theorem 4. *For each clause C_1 and C_2, $\mathcal{P}(mgi(C_1, C_2)) = \mathcal{P}(C_1) \cap \mathcal{P}(C_2)$.*

Proof. Let $e \in \mathcal{P}(mgi(C_1, C_2))$, then according to Definition 11, $mgi(C_1, C_2) \succeq e$. But according to the definition of mgi, $C_1 \succeq mgi(C_1, C_2)$ and $C_2 \succeq mgi(C_1, C_2)$ and therefore $C_1 \succeq e$ and $C_2 \succeq e$. Hence, $e \in \mathcal{P}(C_1)$ and $e \in \mathcal{P}(C_2)$ and therefore $e \in \mathcal{P}(C_1) \cap \mathcal{P}(C_2)$. Hence, $\mathcal{P}(mgi(C_1, C_2)) \subset \mathcal{P}(C_1) \cap \mathcal{P}(C_2)$. Now, let $e \in \mathcal{P}(C_1) \cap \mathcal{P}(C_2)$, then according to Definition 11, $C_1 \succeq e$ and $C_2 \succeq e$. But according to the definition of mgi, $mgi(C_1, C_2) \succeq e$. Hence, $e \in \mathcal{P}(mgi(C_1, C_2))$ and therefore $\mathcal{P}(C_1) \cap \mathcal{P}(C_2) \subset \mathcal{P}(mgi(C_1, C_2))$ and this completes the proof. □

Theorem 5. *For each M, M_1 and M_2 in $\mathcal{M}_n$, $\mathcal{PV}(C(M)) = \mathcal{PV}(C(M_1)) \wedge \mathcal{PV}(C(M_2))$ if $M = M_1 \vee M_2$.*

Proof. Suppose $M = M_1 \vee M_2$. Therefore according to Theorem 3, $C(M) = mgi(C(M_1), C(M_2))$. Then according to Theorem 4 ,$\mathcal{P}(C(M)) = \mathcal{P}(C(M_1)) \cap \mathcal{P}(C(M_2))$. But according to Definition 11, $\mathcal{PV}(C(M)) = \mathcal{PV}(C(M_1)) \wedge \mathcal{PV}(C(M_2))$. □

This theorem, which also holds for negative coverage vectors, can be easily extended for n clauses. According to these theorems, positive (or negative) coverage

[2] Note that the training examples must be prepared for substitution testing with respect to the bottom-clause.

of clauses can be computed by bitwise operations. Hence, the evaluation of each individual is done at genotype level without mapping it into the corresponding phenotype (clause).

Example 4. In Figure 1.b we can compute the coverage vector of any clause provided the coverage vector of individuals 001, 010 and 100. For example: $\mathcal{PV}(C(110)) = \mathcal{PV}(C(100)) \wedge \mathcal{PV}(C(010))$ and $\mathcal{PV}(C(111)) = \mathcal{PV}(C(100)) \wedge \mathcal{PV}(C(010)) \wedge \mathcal{PV}(C(001))$.

4 Stochastic Refinement

According to the theorems in section 2, unification and anti-unification can be done by simple bitwise operations on the binary encoding of clauses. These properties can be used for designing task-specific genetic operators such as generalization and specialization crossover operators. Generalization and specialization are known as the main operations in concept learning methods [28,15,16]. In particular, *lgg* and *mgi* are essential in first-order learning. For example, the ILP system Golem [18] which was successfully applied to a wide range real-world applications [2,4,19] only uses a *lgg* operator which operates on determinacy restricted clauses.

In addition to the generalization and specialization crossovers mentioned earlier, we can also introduce task-specific mutation operators. In the standard mutation operator we use a fixed probability (mutation-rate) for changing 0 and 1 bits[3]. As shown in section 2, the difference between bits in binding matrices determines the substitution order between clauses. Hence, the substitution distance between clauses increases monotonically with the Hamming distance between the corresponding matrices. We can use this property to set different mutation rates for 0 and 1 bits based on a desirable degree of generalization and specialization. This could lead to a directed mutation operator.

This degree of generalization or specialization (which can be also used for crossover operators) is introduced by probabilities for generalizations and specialization. In a genetic search some criteria such as completeness and consistency of current individuals can be used for setting these probabilities [9,6].

In first-order concept learning, upward and downward refinement operators are used for generalization and specialization of clauses [21]. In our case, task-specific genetic operators can be interpreted as *stochastic refinement operators* in the context of first-order concept learning.

5 Implementation and Empirical Evaluation

As shown in the previous sections, the proposed framework has great potential for combining GAs and ILP and could lead to an increased performance in a system which is created based on this framework. In particular, in this section we examine the following two null hypotheses:

[3] A random mutation could result in a matrix which is not consistent with Definition 3. This matrix could be normalized using Definition 3.

INPUT: B=Background knowledge, E=Set of examples
1. If $E = \emptyset$ return B
2. Let e be the first example in E
3. Construct the bottom clause ($\bot$) for e
4. Find the best clause H such that $\square \succeq H \succeq \bot$
5. Let $B = B \cup H$
6. Let $E' = \{e : e \in E, B \models e\}$
7. Let $E = E - E'$
8. Goto 1

Fig. 2. Progol's Covering Algorithm. In the present implementation, the A^*-like search used in step 4 is replaced by a genetic search.

Null hypothesis 1: A GA-ILP system based on the proposed framework does not lead to significantly increased efficiency in any problem involving complex target theories.

Null hypothesis 2: The increased efficiency in Null hypothesis 1 cannot be achieved without substantial decrease of accuracy.

To test these hypotheses, we employed the proposed framework to combine Inverse Entailment in CProgol4.4 with a genetic algorithm. CProgol is an ILP system which develops first-order hypotheses from examples and background knowledge. Figure 2 shows the set covering algorithm used in CProgol. CProgol uses Inverse Entailment [20] to construct the most specific clause (or the bottom-clause) for each example and then searches for the best clause H which subsumes this bottom-clause ($\square \succeq H \succeq \bot$). This introduces a subsumption lattice bounded below by the bottom-clause ($\bot$). The standard CProgol starts from the empty clause ($\square$) and uses an A^*-like algorithm for searching this bounded subsumption lattice (step 4). In the present implementation, this lattice is searched by a genetic search. The details for CProgol's A^*-like search, refinement operator and algorithm for building the bottom-clause can be found in [20].

As shown in section 2, $\mathcal{B}(\bot)$ represents a substitution lattice bounded below by the bottom-clause. We used a genetic algorithm together with the binary encoding for clauses (as described in section 2) to evolve a randomly generated population of binary strings in which each individual corresponds to a member of $\mathcal{B}(\bot)$. Because of simple representation and straightforward operators any standard genetic algorithm can be used for this purpose. We used a *Simple Genetic Algorithm(SGA)* [7] and modified it to suit the representation introduced in this paper. This genetic search evolves a population of hypotheses which all subsume the bottom-clause.

In the following, we explain the material and methods used in our experimentation and then discuss the results.

Material and Methods. In this experiment, we compare the performance of the A^*-like search and the genetic search in learning concepts with different complexities. Figure 3 shows the experimental method used in this experiment.

```
1 for i = 1 to 100 do
2    for j = 1 to 5 do
3       Generate a random target concept T_ij with complexity 5 × j
4       Generate 2 × 100 random training and test examples for concept T_ij
5       Execute Progol on the training examples using the A* search
6          N_ij =number of evaluations before finding hypothesis C_ij
7          A_ij =predictive accuracy of C_ij on the test examples
8       Execute Progol on the training examples using the genetic search
9          N'_ij =number of evaluations before finding hypothesis C'_ij
10         A'_ij =predictive accuracy of C'_ij on the test examples
11   end
12end
13for j = 1 to 5 do
14   Plot average and standard error of N_ij and N'_ij versus 5 × j (i ∈ [1..100])
15for j = 1 to 5 do
16   Plot average and standard error of A_ij and A'_ij versus 5 × j (i ∈ [1..100])
```

Fig. 3. Experimental method.

In this experiment we measure the average performance of the genetic search and the A^* search on 100 different runs. In each run, target concepts with complexities between 5 to 25 are generated. For each target concept, a fixed number (i.e. 100) of examples are generated both for training and testing. After generating random examples, Progol is executed on the training example using the A^* search and the genetic search. For each iteration of the loop the following parameters are recorded: N, the number of evaluations before finding a single clause C which is complete and consistent with respect to the training examples and A, the predictive accuracy of C on the test examples. The average and standard error of these parameters are then plotted against the complexity of the target concepts.

As mentioned before, in this experiment we needed to generate random concepts with a given complexity as well as random training and test example for each concept for measuring the predictive accuracy of the induced hypotheses. For this purpose, we have used a concept generator program in which the concept description language is determined by a Stochastic Logic Program(SLP) [17]. In the present experiment, we have used a concept description language similar to one used in the Michalski's trains problem [14]. Table 1 shows part of the train description language used in this experiment. This program defines the space of all possible carriages. A random train is defined as a list of carriages sampled from this program. In this experiment, the complexity of target concept is measured by the number of specific features which describe the target concept. This is determined by the number of specific carriages in the target train and number of properties for each carriage (see Table 1).

The evaluation function and control parameters which are used by the genetic search are shown in Table 2. The evaluation function uses similar criteria to

Table 1. Part of a stochastic logic program for generating random trains. This program defines the space of all possible carriages. A random train is defined as a list of carriages sampled from this program.

```
carriage(Shape,Length,Double,Roof,Wheels,Load) :-
        shape(Length,Shape),
        double(Length,Shape,Double),
        roof(Length,Shape,Roof),
        wheels(Length,Wheels),
        load(Length,Load).

shape(long,rectangle).  shape(short,rectangle).  shape(short,ellipse).
shape(short,hexagon).   shape(short,u_shaped).   shape(short,bucket).

double(short,rectangle,double).      double(long,rectangle,not_double).
double(short,rectangle,not_double).  double(short,ellipse,not_double).
double(short,hexagon,not_double).    double(short,u_shaped,not_double).
double(short,bucket,not_double).

roof(short,ellipse,arc).     roof(short,hexagon,flat).
roof(long,rectangle,none).   roof(long,rectangle,flat).
roof(long,rectangle,jagged). roof(short,rectangle,none).
roof(short,rectangle,flat).  roof(short,rectangle,peaked).
roof(short,u_shaped,none).   roof(short,u_shaped,flat).
roof(short,u_shaped,peaked). roof(short,bucket,none).
roof(short,bucket,flat).     roof(short,bucket,peaked).

wheels(short,2).  wheels(long,2).  wheels(long,3).

load(short,l(circle,1)).    load(short,l(diamond,1)).
load(short,l(hexagon,1)).   load(short,l(rectangle,1)).
load(short,l(triangle,1)).  load(short,l(utriangle,1)).
load(short,l(circle,2)).    load(short,l(diamond,2)).
load(short,l(hexagon,2)).   load(short,l(rectangle,2)).
load(short,l(triangle,2)).  load(short,l(utriangle,2)).
load(long,l(circle,1)).     load(long,l(diamond,1)).
load(long,l(hexagon,1)).    load(long,l(rectangle,1)).
load(long,l(triangle,1)).   load(long,l(utriangle,1)).
load(long,l(circle,2)).     load(long,l(diamond,2)).
load(long,l(hexagon,2)).    load(long,l(rectangle,2)).
load(long,l(triangle,2)).   load(long,l(utriangle,2)).
load(long,l(circle,3)).     load(long,l(diamond,3)).
load(long,l(hexagon,3)).    load(long,l(rectangle,3)).
load(long,l(triangle,3)).   load(long,l(utriangle,3)).
```

Table 2. Control parameters and evaluation function used by the genetic search in the present experimentation.

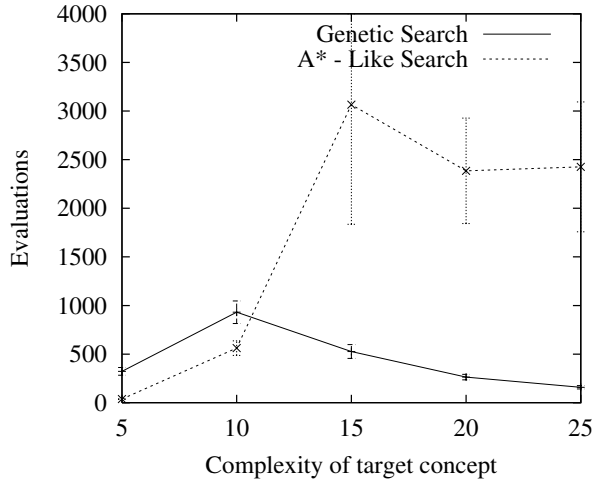
Population size (*popsize*): 30
Probability of mutation (p_m): 0.0333
Probability of crossover (p_c): $1 - 0.8 \times f$
Probability of *lgg* (p_{lgg}): $0.8 \times f$
($f = \frac{f(C_1)+f(C_2)}{2}$ where C1 and C2 are parental clauses in crossover and lgg)
Evaluation function: $f(C) = \alpha \frac{e^+(C)}{(E^+ + \beta * e^-(C))} + (1-\alpha)(1 - \frac{c(C)+h(C)}{c_{max}+h_{max}})$
where $\alpha = 0.8$ and $\beta = 0.5$
E^+, E^-: total number of positive and negative examples
$e^+(C), e^-(C)$: number of positive and negative examples covered by clause C
$c(C)$: length of clause C
$h(C)$: number of further literals to complete clause C (as defined in [20])

the ones used in the evaluation function of the A^*-like search of CProgol. The probability setting for generalisation used in this experiment is similar to one used in [6].

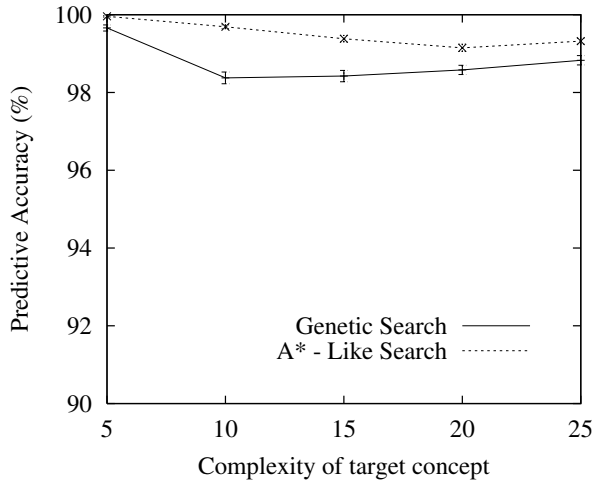
Results and Discussion. Figure 4.a compares the average number of clauses evaluated by the genetic search and the A^*-like search in learning concepts with different complexities. The vertical axis shows the average number of the explored nodes before finding a complete and consistent hypothesis. The horizontal axis shows the complexity of target concept which is measured by the number of conditions in the target concept. This figure shows that in this experiment the A^*-like search exhibits a better performance in learning concepts with small complexities (e.g. less than 10). However, the number of evaluations by the A^*-like search grows very sharply in the complexity of target concept and varies between 38 and 3065.

Figure 4.b compares the predictive accuracy of the induced hypotheses by the genetic search and the A^*-like search. This figure shows that the predictive accuracy of the genetic search is slightly lower (less than 2%) than the predictive accuracy of the A^*-like search.

In summary, these graphs suggest that in the random trains problem the genetic search can lead to significantly increased efficiency for learning complex target trains and this can be achieved without substantial decrease of accuracy. These graphs also suggest that the performance of the genetic search is less dependent on the complexity of hypotheses, whereas A^*-like search shows a great dependency on this factor. The results of this experiment are consistent with the fact that the actual completeness and complexity exhibited by the standard A^*-like search of CProgol depends upon the order of literals in the bottom clause and upon the complexity of the hypothesis. In contrast, the genetic search is less dependent on the complexity of the hypotheses and is not affected by the order of literals in the bottom clause.



(a)



(b)

Fig. 4. Performance of the genetic search and A^*-like search in learning concepts with different complexities. a) average number of clauses evaluated by each search method and b) predictive accuracy of the induced hypotheses versus the complexity of concepts.

6 Related Work

One main difficulty in applying conventional GAs in first-order domain is related to the formulation of first-order hypotheses into bit-strings. GA-SMART [6] was

the first relation learning system which tackled this problem by restricting concept description language and introducing a language template. A template in GA-SMART is a fixed length CNF formula which must be defined by the user. Mapping a formula into bit-string is done by setting the corresponding bits to represent the occurrences of predicates in the formula. The main problem of this method is that the number of conjuncts in the template grows combinatorially with the number of predicates. REGAL [5], DOGMA [8] and G-NET [1] follow the same basic idea as GA-SMART and employ a user-defined template for mapping first-order rules into bit strings. However, instead of using a standard representation, a template in these systems is a conjunction of internally disjunctive predicates. This leads to some difficulties, for example in representing continuous attributes. Other systems including GILP [27], GLPS [13], LOGENPRO [29], STEPS [11] and EVIL [23] use hierarchical representations rather than fixed length bit-strings. These systems evolve a population of logic programs in a Genetic Programming (GP) [12] manner. Even though some of the above mentioned systems use background knowledge for generating the initial population or seeding the population, most of these systems cannot benefit from intentional background knowledge in the same way as in usual first-order learning systems. This is mainly because in most cases genetic search has been used as the only learning mechanism in the system.

In our proposed framework, encoding of hypotheses is based on a most specific (or bottom) clause which is constructed according to the background knowledge and training examples. This bottom-clause can be automatically constructed using logic-based methods such as Inverse Entailment. Moreover, as shown in section 2 and section 3, the proposed encoding and operators can be interpreted in well known first-order logic terms.

7 Conclusions and Further Work

In this paper we have introduced a framework for combining first-order concept learning with GAs by introducing a novel encoding for clauses, relevant genetic operators and a fast evaluation mechanism. Empirical results suggest that the proposed framework could lead to significantly increased efficiency in problems involving complex target theories and this can be achieved without substantial decrease of accuracy.

The present implementation could be improved in many ways. A natural improvement might be using more sophisticated genetic algorithms rather than a simple genetic algorithm. For example the greedy cover set algorithm of CProgol, which repeatedly generalizes examples, could be replaced by a parallel genetic algorithm. The task-specific genetic operators can be used to guide the genetic search towards the interesting areas of the search space by specialization and/or generalization as it is done in usual concept learning systems. The fast evaluation mechanism can be used to compensate for the natural computation cost of a genetic algorithm and could lead to a high performance genetic search. In the current approach, the occurrence of atoms in a clause is not considered in the binary encoding of the clause and inactive atoms (e.g. unconnected predicates)

are filtered from the induced hypotheses. Alternatively, the presence or absence of atoms in each clause can be encoded as a part of the binary representation of the clause. Finally, more experiments are required to evaluate the proposed framework in real-world domains.

The framework proposed in this paper is an opportunity not only to utilize the benefits of these two different paradigms (i.e. ILP and GAs) in a hybrid system but also to study some common issues with an analogical view. In the following we review some of these issues which could be considered as further research.

Hybrid Search. It has been argued [7] that GAs are a weak method without the guarantee of optimality. In other words, GAs sort out interesting area of a space quickly without the guarantees of more convergent process. In ILP problems, which known local but convergent methods exist, the idea of a hybrid GA [3] is natural. In this scheme the search is started using a GA to sort out the interesting hills in the problem. Once the GA locates the best regions, a locally convergent search is used to climb the local peaks. In this way, one can combine the globality and parallelism of the GA with the more convergent behaviour of the local search (i.e. the ILP techniques). In this hybrid scheme, the GA performs a global adaptive search of the space of possible hypotheses and then an ILP algorithm locally refines an initial estimate provided by the GA. Theory revision in ILP could be relevant in designing a local search in this scheme. This hybrid scheme also provides an opportunity to study the interaction between the computational models of 'evolution' and 'learning'.

Parallel Search. In addition to the well-known implicit parallelism[4], GAs are naturally suitable for parallel implementation [25]. This makes it easier to scale-up GA-based systems and to benefit from the computational power of parallel and distributed hardware. Hence, in a GA-based ILP system a reasonable attempt is to parallelize the search. Moreover, there are some situations in ILP methods where parallel processing could be useful. For example, the greedy set covering algorithm which repeatedly generalizes training examples, could be replaced by a parallel co-evolutionary procedure [1].

Learning Clausal Theories. In the current approach each individual in the population stands for a single clause. The final solution consists of clauses each corresponding to an iteration of the cover set algorithm. An alternative is to induce the whole clausal theory at once. In a GA-based system this requires encoding of clausal theories as bit-strings which are regarded as individuals in the population. This problem is similar to Pittsburgh approach [24,10] in which each individual encodes a set of production rules. To be able to learn clausal theories, the proposed binary encoding must be extended to represent multiple clauses.

[4] It has been shown [7] that even though in each generation we perform computation proportional to the size of the population (n) we get useful processing of much more schemata ($O(n^3)$) in parallel with no special bookkeeping or memory other than the population itself.

Acknowledgements

We would like to thank the anonymous reviewers for their comments and constructive criticism. The first author was supported by an ILPnet2 travel funding to attend ILP02 conference.

References

1. C. Anglano, A. Giordana, G. Lo Bello, and L. Saitta. An experimental evaluation of coevolutive concept learning. pages 19–27. Morgan Kaufmann, 1998.
2. I. Bratko, S. Muggleton, and A. Varsek. Learning qualitative models of dynamic systems. In *Proceedings of the Eighth International Machine Learning Workshop*, San Mateo, Ca, 1991. Morgan-Kaufmann.
3. L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.
4. C. Feng. Inducing temporal fault dignostic rules from a qualitative model. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, London, 1992.
5. A. Giordana and F. Neri. Search-intensive concept induction. *Evolutionary Computation Journal*, 3(4):375–416, 1996.
6. A. Giordana and C. Sale. Learning structured concepts using genetic algorithms. pages 169–178. Morgan Kaufmann, 1992.
7. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, Reading, MA, 1989.
8. J. Hekanaho. Dogma: A ga-based relational learner. pages 205–214. Springer-Verlag, 1998.
9. C. Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13:189–228, 1993.
10. Kenneth A. De Jong, William M. Spears, and Diana F. Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 13:161–188, 1993.
11. C. J. Kennedy and C. Giraud-Carrier. An evolutionary approach to concept learning with structured data. In *Proceedings of the fourth International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 1–6. Springer Verlag, April 1999.
12. J. R. Koza. *Genetic Programming*. MIT Press, Cambridge, MA, 1991.
13. K. S. Leung and M. L. Wong. Genetic logic programming and applications. *IEEE Expert*, 10(5):68–76, 1995.
14. R.S. Michalski. Pattern recognition as rule-guided inductive inference. In *Proceedings of IEEE Trans. on Pattern Analysis and Machine Intelligence*, pages 349–361, 1980.
15. T.M. Mitchell. Generalisation as search. *Artificial Intelligence*, 18:203–226, 1982.
16. T. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
17. S.H. Muggleton. Stochastic logic programs. In L. de Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, 1996.
18. S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, Tokyo, 1990. Ohmsha.
19. S. Muggleton, R. King, and M. Sternberg. Protein secondary structure prediction using logic-based machine learning. *Protein Engineering*, 5(7):647–657, 1992.
20. S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

21. S-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*. Springer-Verlag, Berlin, 1997. LNAI 1228.
22. G.D. Plotkin. A note on inductive generalisation. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pages 153–163. Edinburgh University Press, Edinburgh, 1969.
23. P. G. K. Reiser and P. J. Riddle. Evolving logic programs to classify chess-endgame positions. In C. Newton, editor, *Second Asia-Pacific Conference on Simulated Evolution and Learning*, Canberra, Australia, 1998.
24. S F Smith. Flexible learning of problem solving heuristics through adaptive search. In *Proc. 8th Int. Joint Conf. on A.I.*, pages 422–425, 1983.
25. J. Stender. *Parallel Genetic Algorithms: Theory and Practice*. IOS Press, Amsterdam, 1993.
26. A. Tamaddoni-Nezhad and S. H. Muggleton. Searching the subsumption lattice by a genetic algorithm. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, pages 243–252. Springer-Verlag, 2000.
27. A. Varšek. *Inductive Logic Programming with Genetic Algorithms*. PhD thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Ljubljana, Slovenia, 1993.
28. P. H. Winston. *Learning Structural Descriptions from Examples*. Phd thesis, MIT, Cambridge, Massachusetts, January 1970.
29. M. L. Wong and K. S. Leung. Evolutionary program induction directed by logic grammars. *Evolutionary Computation*, 5(2):143–180, 1997.

Experimental Investigation of Pruning Methods for Relational Pattern Discovery

Irene Weber

Institut für Informatik, Universität Stuttgart
Breitwiesenstr. 20–22, 70565 Stuttgart, Germany
Irene.Weber@informatik.uni-stuttgart.de

Abstract. Finding all interesting patterns in a database is a data mining task that typically requires a complete search through the hypothesis space. Several ILP systems address this task, e.g., [Deh98,Wro97,FL01]. Safe pruning techniques that reduce the size of the hypothesis space without the risk of missing interesting patterns are very important for this task. This paper is concerned with the effectiveness of pruning techniques in this setting. The addressed pruning techniques are (1) optimum estimates, (2) a pruning technique based on subset tests that is derived from the Apriori search algorithm, (3) pruning based on taxonomies, and (4) to consider only most general patterns as interesting. Methods (1) to (3) are safe pruning techniques that find all interesting patterns; method (4) reduces the number of accepted patterns. The effect of these pruning methods is investigated by experiments within a range of different specific task settings and two databases.
Experimental results indicate that optimum estimates and Apriori-style pruning are effective and reliable pruning techniques that produce little additional cost. The effect of taxonomies for pruning is smaller, and it varies over different task settings. In the experiments, the restriction to most general patterns considerably reduces the search costs as well as the set of accepted patterns.

1 Introduction

Knowledge discovery and data mining are concerned with the discovery of valid, novel, potentially useful, and understandable patterns in data. ILP methods are particularly interesting for data mining because they are applicable directly to multi-relational databases. In contrast, most other algorithms require that the data be transformed into a single, attribute-value table. However, in general, it is not sensible to convert databases with non-determinate or recursive links between relations into attribute-value format since the transformation blows up the table size and generates many redundant or unknown attribute values.

Finding all interesting patterns in databases is an important data mining task. It is defined as follows: Given a database D, a pattern language L and an acceptance criterion $q : (L, D) \rightarrow \{true, \text{false}\}$, find all patterns $p \in L$ where $q(p, D)$ is true. As the task definition demands that all patterns that are interesting in terms of the acceptance criterion are discovered, it does not allow

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 301–316, 2003.

for heuristic pruning, and pattern discovery algorithms have to conduct a complete search through the pattern space. Due to the expressivity of first-order languages, ILP search spaces typically are very large. Therefore, safe pruning techniques that prune the search space without losing acceptable patterns are crucial for ILP pattern discovery systems.

This paper is concerned with the effectiveness of pruning techniques that are applicable for pattern discovery within a restricted ILP setting where all patterns describe subgroups of a fixed population of individuals. Several systems have been developed for this task in the framework of ILP, e.g., Warmr, Midos, and Tertius [Deh98,Wro97,FL01].

The pruning techniques are (1) optimum estimates, (2) a pruning technique based on subset tests that is derived from the Apriori search algorithm, and (3) pruning based on taxonomies. These pruning techniques are safe, that is, they detect all acceptable patterns that are detected by a complete search not pruning the search space. The effect of these pruning strategies is investigated by experiments in various specific task settings with two databases, one of which contains non-determinate links, and thus is a true ILP database. A further means to restrict the search space is (4) to define only most general patterns as interesting. Whenever a pattern is a accepted, the patterns it subsumes can be pruned from the search space. Obviously, the restriction to most general patterns cannot be considered a safe pruning technique since it modifies the acceptance criterion such that fewer patterns are accepted. It is investigated by further experiments whether the resulting loss of accepted patterns is rewarded by a reduction of the search costs.

The next section defines the formal framework of pattern discovery assumed here. The third section introduces the basic search algorithm and elaborates on the pruning techniques that are investigated. The experimental setup, mainly the specific tasks and the databases, are described in section 4. Section 5 reports and discusses the results of the experiments.

2 Subgroup Discovery in the Framework of Descriptive ILP

The task of finding interesting patterns addressed here assumes that the patterns describe subgroups of a fixed population of individuals. It is similar to the settings of [Deh98,FL01,Wro97]. The individual population is defined as the set of distinct bindings of specified individual variables in the database at hand.

Definition 1. *The population P is the set of all ground substitutions θ of specified variables $\{V_1, \ldots, V_n\}$ occurring in the population literal pop for which $pop\theta$ is true in the database D, i. e., $P = \{(V_1 \ldots, V_n)\theta \mid D \models pop\theta$ and $(V_1 \ldots, V_n)\theta$ is ground$\}$.*

A pattern p is a logical expression that has to share variables with the population literal *pop*. The form and construction of p are explained below.

Definition 2. *The coverage of a pattern p is the set of individuals $(V_1, \ldots, V_n)\theta$ for which $D \models (pop \wedge p)\theta$. The coverage of p is denoted $cov(p)$.*

The interestingness of patterns is evaluated with respect to a target group. The target group is a subset of the population that is in the focus of interest in the given application. T is defined by a logical expression t.

Definition 3. *The target group T is the subset of the population for which the logical expression t is true, i. e., $T = \{(V_1 \ldots, V_n)\theta \mid D \models (pop \wedge t)\theta$ and $(V_1 \ldots, V_n)\theta$ is ground$\}$.*

The rating of the interestingness of a pattern is computed from the size of the coverage of a pattern, $|cov(p)|$ and the number of target objects the pattern covers, $|cov(p \wedge t)|$. The set of target objects covered by a pattern p is termed the supporting set of p. The set of non-target objects covered by a pattern p is termed the contradicting set of p. The sizes of these sets are computed by appropriate database queries. A data mining system using Prolog for data storage and access can compute the sizes of the coverage $C = cov(p)$ and supporting set $S = cov(p \wedge t)$ of a pattern p by the following queries:

$$findall((V_1, \ldots, V_n), (pop, p), C), length(C, SC).$$
$$findall((V_1, \ldots, V_n), (pop, p, t), S), length(S, SS).$$

3 The Basic Search Algorithm

Pattern Language. Patterns are conjuncts of so-called s-literals and are represented as sets of s-literals. S-literals are similar to first-order features, they are declared manually. The set S of s-literals that can occur in patterns is predefined and fixed. Patterns are subsets of S. From the logical point of view, an s-literal is a conjunct of first-order literals, or a first-order expression that – in the context of a pattern — can be evaluated by a Prolog or SLDNF proof procedure.

In order to prevent contradicting or uninteresting combinations of s-literals in patterns, an excludes-list $excl(l)$ and a requires-list $req(l)$ can be provided for an s-literal l. The excludes-list $excl(l)$ of an s-literal l is a list of s-literals that must not co-occur with l in a pattern, whereas the s-literals in the requires-list $req(l)$ of l are required to occur in any pattern that contains l. The pattern language is restricted to such sets of s-literals that satisfy the requires- and excludes constraints. Examples for s-literals are shown in section 5.2.

Specialization Operator. The task definition of interesting pattern discovery demands that all patterns that meet the acceptance criterion are discovered. Therefore, a complete search through the pattern language is required. In ILP, the search for patterns is commonly organized as the systematic, repeated application of a refinement operator. In top-down search, the refinement operator starts with the most general pattern in the pattern language. Patterns are specialized, and the resulting patterns are refined further, until the search space

is traversed completely. In the pattern language used here, a pattern p is specialized by adding another s-literal l to p. The specialization operator obeys the restrictions expressed by excludes- and requires-lists, i.e, it adds an s-literal l only to a pattern p if $req(l) \subset p$ and $excl(l) \cap p = \emptyset$.

Generally, there are several ways to generate a pattern. In order to avoid redundancy, search algorithms take care to restrict the application of specialization operators such that, for any pattern p exactly one sequence of specialization steps producing p is allowed. For the specialization operator used here, an order in which s-literals have to occur in a pattern is fixed.

Basic Pruning. Acceptance criteria typically are derived from numerical interestingness functions for patterns. Patterns are defined as acceptable if their interestingness reaches or exceeds a certain threshold. Specializing a pattern p produces patterns p' that are subsumed by p. A pattern p is said to subsume a pattern p' (written $p \succ p'$), if $cov(p) \supseteq cov(p')$.

For interestingness functions that are monotonous to the size of the coverage of a pattern, specializing a non-acceptable pattern will always produce non-acceptable patterns because specialization of a pattern produces patterns with smaller or equal coverage size. Consequently, non-acceptable patterns need not be specialized, but can be pruned from the search space. Thus, the specialization operator implements a basic form of safe pruning. It is realized in most ILP pattern discovery systems.

4 Pruning Methods

Complete Pruning by Subset Condition. Searching and pruning with a specialization operator as sketched so far has a weakness. If the search algorithm detects a pattern p with insufficient coverage, it ceases further specialization of the pattern p. Thus, it prunes only such patterns from the search space, that are generated by a sequence of specialization operations including the pattern p. Other patterns that are subsumed by p, but generated by specialization sequences not including p remain in the search space.

The well-known Apriori algorithm [AMS$^+$96] that was developed for finding associations rules in transaction databases implements a pruning strategy that exploits all subsumption relationships between patterns for pruning. The first step of finding association rules is the search for frequent associations of items in the transactions. This task is an instance of finding all interesting patterns in a database where the patterns are associations of items, and the acceptance criterion is frequency. A pattern is frequent if the size of its coverage reaches a certain threshold. The coverage of a pattern is the number of transactions in the given database which contain the pattern as a subset.

The pattern language of this application is very simple. A pattern is a subset of a fixed set of items. All subsets of the item sets belong to the pattern language. The core idea of the Apriori search algorithm is based on the observation that if a given pattern p is frequent, all patterns that are subsets of p are frequent as well. The Apriori search algorithm generates and evaluates patterns from general

to specific, and stores the frequent patterns, i.e., the ones that are worth further specialization, in a set F. For any newly generated pattern p, the algorithm checks the *subset condition*. The subset condition is true for a pattern p if all subsets of p that are one item shorter than p, occur in F. If the subset condition is not true for a pattern, the pattern is subsumed by an infrequent pattern, and therefore, it is infrequent as well.

In order to avoid expensive θ-subsumption tests, we adopt the Apriori search strategy and subset condition for an ILP language. The s-literals play the role of the items in the original Apriori search algorithm. Essentially, the ILP pattern language differs from the original Apriori pattern language in the excludes- and requires-lists. They allow to restrict the pattern language to meaningful combinations of s-literals. The subset condition has to be modified such that it only checks occurrence of patterns of the restricted pattern language in the set F of patterns stored for further specialization. The approach is introduced and described in more detail in [Web00].

Optimum Estimate Functions. Many practically relevant interestingness functions are not monotonous to coverage size, e.g., interestingness functions based on statistical tests. For some interestingness functions, optimum estimate functions can be derived. Given a pattern p, an optimum estimate function for a interestingness function f computes an upper bound for the interestingness values $f(p')$ that patterns p' that are subsumed by p can reach. If the upper bound is smaller than the threshold defined by the acceptance criterion, specialization of pattern p can be stopped and its specializations can be pruned. Pruning based on optimum estimates is safe for correct optimum estimates that compute true upper bounds.

Exploiting Subsumption Relationships between s-Literals for Pruning. An s-literal l subsumes an s-literal s' (written $l \succ l'$) iff $cov(pop \wedge l) \supseteq cov(pop \wedge l')$. Subsumption relationships between s-literals are caused, e.g., by taxonomies on attribute domains. For example, in a database modeling a blocks world that includes a relation $objects(OId, Colour, Shape, Size)$, the s-literal $Shape = square$ is more special than the s-literal $Shape = rectangle$. Subsumption relationships between s-literals induce subsumption relationships between patterns: if a pattern p contains an s-literal l with $l \succ l'$, and l is replaced by l' in p, the resulting pattern $p' = (p \setminus \{l\}) \cup \{l'\}$ is subsumed by p, i.e., $p \succ p'$. These subsumption relationships offer an opportunity to prune the search space. Pruning based on subsumption relationships between s-literals can be integrated with pruning based on the subset condition if patterns are represented in *genex* form. A pattern p is in *genex* form iff for all $l \in p$ the following holds: if there is an s-literal l' declared for the pattern language with $l' \succ l$, then $l' \in p$. The *genex* representation of a pattern is unique, i.e., it does not allow syntactical variants.

The specialization operator is forced to generate patterns in *genex* form if for every subsumption relationship $l \succ l'$ a requires constraint is declared such that

the subsumed literal l' requires the more general literal l, i. e., $l \in req(l')$. Given the appropriate *requires* constraints, pruning based on subsumption relationships between s-literals is realized along with subset pruning and does not require extra procedures.

Restriction to Most General Patterns. The pruning methods described so far realize safe pruning, i.e., they do not change the set of accepted patterns. Restriction to most general patterns means that only most general patterns are accepted. Specializations of accepted patterns are pruned from the search space. This is realized by treating accepted patterns similar to patterns that can be pruned due to insufficient coverage or optimum estimate. Restriction to most general patterns is not a safe pruning technique, but rather implements a different acceptance criterion.

5 Experimental Setup

5.1 Task Settings

Three data mining task settings with different interestingness functions and acceptance criteria are specified for the experiments.

Assoc. The first task setting, referred to as *assoc*, uses two interestingness functions, *frequency* and *confidence*. They are defined as follows.

Definition 4. *The frequency supp of pattern p wrt. to a target group t in a database D is* $supp(p) = \frac{|cov(p \wedge t)|}{|cov(t)|}$.

The confidence conf of pattern p wrt. to a target group t in a database D is $conf(p) = \frac{|cov(p \wedge t)|}{|cov(p)|}$.

Obviously, the support of a pattern is monotonous to the coverage, and allows to prune the search space accordingly. The confidence is not monotonous to the coverage. A non-trivial optimum estimate for confidence does not exist, since there is always the chance of a specialization step that excludes exactly the contradicting individuals from the coverage of the pattern, yielding a pattern with maximum confidence 1.

Midos. The second task setting is called midos. Its interestingness function is distributional unusualness combined with a frequency threshold σ_m. It was adapted for relational data mining by Wrobel [Wro97]

Definition 5. *The distributional unusualness du of a pattern p is defined as*

$$du(p) = \begin{cases} -1 & \text{if } \frac{|cov(p)|}{|P|} < \sigma_m \\ \frac{|cov(p)|}{|P|} \cdot \left(\frac{|cov(p \wedge t)|}{|cov(p)|} - \frac{|cov(t)|}{|P|} \right) & \text{else} \end{cases}$$

The task is to find the k most interesting patterns wrt. distributional unusualness with sufficient frequency. A pattern p is accepted if $\frac{|cov(p)|}{|P|} \geq \sigma_m$ and $du(p) > min\{du(h) \mid h \in H\}$ where H is the set of accepted patterns. When a pattern p is accepted, it is added to H if $|H| < k$. If $|H| = k$, a new pattern p is accepted by substituting it to the h with minimal $du(h)$ in H.

The first branch of the definition of du allows to prune patterns with too small coverage. Additionally, there are three optimum estimate functions for the second branch of the definition of distributional unusualness.

$$du_{oe1}(p) = \frac{|cov(p \wedge t)|}{|P|} \cdot \left(1 - \frac{|cov(t)|}{|P|}\right)$$
$$du_{oe2}(p) = \frac{|cov(p \wedge t)|}{|P|} - \sigma_m \cdot \frac{|T|}{|P|}$$
$$du_{oe3}(p) = \frac{|cov(p)|}{|P|} \cdot \left(1 - \frac{|cov(t)|}{|P|}\right)$$

Optimum estimate du_{oe1} expresses the observation that a most favorable specialization operation of a pattern p excludes all and only the non-target individuals from $cov(p)$. Optimum estimate du_{oe2} is based on the idea that a favorable specialization excludes as much non-target objects as can be excluded without violating the minimum frequency constraint $cov(p) \geq \sigma_m \cdot |P|$. Optimum estimate du_{oe3} is a weaker version of du_{oe1} that is used in experiments on the effect of optimum estimates.

Impint. The interestingness function of the third task setting impint is implication intensity II. Implication intensity was developed by [GL93] and investigated as a measure of interest for KDD by [FDPB95]. It is a statistical measure for the strength of the implication expressed by a rule $A \rightarrow B$. Here, it is used to rate how strong the fact that an individual i is covered by a pattern p implies that i belongs to the target group T, i.e., the intensity of the implication $i \in cov(p) \rightarrow i \in T$.

Definition 6. *Let $\overline{T} = P \setminus cov(t)$, $\lambda = |cov(p)| \cdot \frac{|\overline{T}|}{|P|}$, and Φ the cumulated standard normal distribution. The implication intensity II of a pattern p wrt. a target group $cov(t)$ is*

$$II(p) = \begin{cases} 1 - \sum_{i=0}^{|cov(p) \cap \overline{T}|} \frac{\lambda^i}{i!} \cdot \mathbf{e}^{-\lambda} & if \lambda \leq 3 \\ 1 - \Phi\left(\frac{|cov(p) \cap \overline{T}| - \lambda}{\sqrt{\lambda}}\right) & else \end{cases}$$

The first branch of the definition computes the implication intensity based on the Poisson distribution, the second branch approximates the Poisson distribution by the normal distribution which is sufficiently exact for $\lambda \geq 3$ [FDPB95].

A pattern p is accepted if $II(p)$ reaches a threshold $1 - \alpha$ and if its coverage $cov(p)$ meets a minimum frequency threshold σ_i, i.e., the following must hold for any acceptable pattern p.

$$II(p) \geq 1 - \alpha$$
$$\frac{cov(p)}{|P|} \geq \sigma_i$$

In the impint setting, acceptance is restricted to most general patterns; accepted patterns are excluded from further specialization.

The threshold σ_i is chosen such that $\sigma_i > \frac{3}{|\overline{T}|}$. Thus, the normal distribution branch of II applies for all acceptable patterns. There are two optimum estimates for the normal distribution branch of implication intensity [Web00].

$$II_{oe1}(p) = 1 - \Phi\left(-\sqrt{cov(p) - \big(cov(p) \cap \overline{T}\big)} \cdot \frac{\sqrt{\overline{T}}}{\sqrt{|P|}}\right)$$

II_{oe1} uses the same principle as the optimum estimate du_{oe1} for distributional unusualness, namely, that a most favorable specialization of pattern l would exclude exactly the contradicting instances $cov(p) \cap \overline{T}$ from its coverage $cov(p)$.

II_{oe2} is an improvement of II_{oe1}. It uses the fact that a most favorable specialization operation must keep enough contradicting individuals to meet the minimum frequency constraint. For $\text{mincov} = \lceil \sigma_i \cdot |P| \rceil$ the minimum size of the coverage of a pattern that is required by the minimum frequency constraint, and $\lambda = mincov \cdot \frac{|\overline{T}|}{|P|}$,

$$II_{oe2}(p) = 1 - \Phi\left(\frac{(mincov - cov(p)) - \lambda}{\sqrt{\lambda}}\right)$$

5.2 Databases and Search Spaces

The experiments were conducted with two databases, namely the well-known KRK database describing chess boards of the KRK end game, and a phonetics database. For each database, a search space is defined in two variants. The pattern languages of both variants are identical, i.e., both search spaces include the same patterns. The difference is that the tax variant models taxonomies and generalization relationships between s-literals, whereas the no_tax variant does not. Consequently, the no_tax variant does not offer the possibility of pruning based on taxonomies and generalization relationships between s-literals. For the assoc and midos task setting, the search through the tax and the no_tax variant of a search space discovers the same set of acceptable patterns. For the impint setting, this is not the case because the restriction to most general patterns interacts with the taxonomy information. The taxonomy induces subsumption relationships between patterns that allow to prune the subsumed patterns due to the restriction to most general patterns. In the no_tax variant of the search space, these subsumption relationships are not known, and more patterns have to be accepted.

The KRK Database. The KRK chess dataset[1] represents chess board positions in the KRK endgame with three pieces left on the board: White King (*WK*), White Rook (*WR*), and Black King (*BK*). It contains 20,000 facts of the form $krk(Id, Class, WKC, WKR, WRC, WRR, BKC, BKR)$. The variable Id stands for identifiers of board positions, $Class$ takes the value 1 for illegal board positions and 0 for legal board positions. *WKC* is the column position of *WK*, *WKR* is the row position of *WK* and so on. Column and row positions are represented as integers ranging from 0 to 7. The population is defined as (distinct) ground instantiations of the variable Id in the literal $krk(Id, Class, WKC, \ldots)$. The literal $Class = 1$ identifies target objects.

Thirty s-literals are defined that compare row positions or column positions of the chess pieces. E.g., the following s-literals refer to the column positions of *WK* and *BK*: "$WKC = BKC$", "$WKC \neq BKC$", "$WKC > BKC$", "$WKC < BKC$", and "$N1$ *is* $WKC - BKC, (N1 = 1; N1 = -1; N1 = 0)$". The last literal defines $adjacent(WKC, BKC)$.

For the tax variant, the system was provided with the information that literals of the form $X \neq Y$ are more general than literals of the form $X < Y$ and $X > Y$ and that literals $adjacent(X, Y)$ are more general than literals $X = Y$.

The Phonetics Database. The phonetics database provides scientific data on German speech recordings, collected and prepared for scientific phonetics analysis [Rap98]. The database consists of three relations *syllables* with 32 attributes and 18109 tuples, *phonemes* with 10 attributes and 48093 tuples, and *words* with 7 attributes and 7867 tuples. In order to limit the size of the pattern language, s-literals are defined only for a selection of the available attributes. The abbreviated database schemas are as follows:

$$syllables(SId, WId, LExp, LMes, D2P, TAlign, PHeight, Acc)$$
$$phonemes(PId, SId, Phoneme, Type)$$
$$words(WId, Tag)$$

The attributes *SId*, *WId* and *PId* identify instances of syllables, words, and phonemes. They are primary keys of the respective relations. The experiments investigate the population of syllables, identified by the key attribute *SId* of relation *syllables*. The population literal is $syllables(SId, WId, \ldots, Acc)$. The attribute *Acc* has value 1 if the respective syllable has a pitch accent, otherwise, its value is 0. The target group is defined by "$Acc = 1$".

The attributes *LExp*, *LMes*, *D2P*, *TAlign*, *PHeight* are continuous numerical attributes of the spoken syllables. Eight s-literals compare these numerical attributes of a syllable with expected values or with the average value of syllables in that attribute, e. g., "$LMes < LExp$".

The link between syllables and words is established via the variable *WId*. The tag marks the function of the word in the context of the sentence, e.g., noun, verb, etc. For tags, a taxonomy is available, with maximum depth 3 and maximum

[1] http://www-users.cs.york.ac.uk/~stephen/chess.html

branching factor 9. There are 40 basic tags, as defined in the Stuttgart-Tübingen-Tag-Set [Rap98], and 14 generalized tags. E.g., tag adja (attributives Adjektiv) and tag adjd (adverbiales or prädikatives Adjektiv) generalize to adj (Adjektiv). Correspondingly, 54 s-literals are defined, e.g., "$word(WId, Tag), Tag = adja$", "$word(WId, Tag), Tag = adjd$", or "$word(WId, Tag), (Tag = adja; Tag = adjd)$" for generalized tags. As a syllable belongs to exactly one word, and a word has exactly one tag, the language is restricted such that, in each pattern, at most one s-literal concerning tags is allowed in the no_tax variant of the search space. The tax variant allows co-occurrences of such patterns only for genex representation.

Further s-literals refer to the kind and type of the phonemes of a syllable. There are 39 distinct kinds of phonemes and 8 types that generalize the phonemes. The link between syllables and phonemes, that is defined via the variable *SId*, is non-determinate, since a syllable, in general, consists of several phonemes. Patterns can describe combinations of phoneme kinds or phoneme types in syllables. For instance, the following pattern describes syllables containing a short vowel and a fricative:

$$syllables(SId, WId, \dots), phonemes(PId1, SId, P1, T1),$$
$$T1 = short, phonemes(PId2, SId, P2, T2), T2 = fric.$$

5.3 Search Runs

Four series of search runs are conducted in each task setting. A first series of search runs (all) exploits all available possibilities to prune the search space. In the assoc setting, the subset condition and subsumption relationships between pairs of s-literals are exploited for pruning. In the midos and impint settings, additionally, the best available optimum estimates were computed, i. e., du_{oe1} and du_{oe2} in the midos setting, and II_{oe2} in the impint setting.

The second series evaluates the contribution of optimum estimates to search cost reduction. As the assoc setting does not offer any optimum estimates, no search runs for assoc are conducted in this series.

The third series of search runs searches the no_tax version of search spaces and, thus, makes no use of subsumption relationships between pairs of s-literals for pruning.

The fourth series evaluates the reduction of search efforts gained by the restriction to most general patterns. These search runs solve a modified impint task setting where specializations of accepted patterns remain in the search space.

Each series consists of three search runs. The search runs were executed with different settings of a parameter of the acceptance criterion. In the assoc task setting, the acceptance criterion accepts all patterns with minimum confidence 0.8 and minimum frequency σ_a where σ_a takes values $0.1, 0.2, 0.3$ and 0.4 in different search runs. The confidence threshold influences the number of acceptable patterns, but has no effect on pruning. Its is set to 0.8 for all search runs. In the midos setting, the parameter σ_m is $0.0001, 0.1, 0.2$ and 0.3 in different

search runs. The parameter k is 10 in all search runs. In the impint setting, the parameter σ_i is fixed to $S = 0.005$ in all search runs. The parameter α takes values $0.0001, 0.005, 0.01, 0.05$. The parameter settings vary the acceptance criterion and, consequently, the number of acceptable patterns as well as the extent of pruning that is possible. In the assoc and midos settings, the acceptance criterion grows stricter with increasing σ_a and σ_i, respectively. In the impint setting, the acceptance criterion also works as a pruning criterion: due to the restriction to most general patterns (rgen), accepting a pattern allows to prune its specializations. With a tight acceptance criterion (i.e., small α), less pruning based on rgen and more pruning by the other methods (subset condition, optimum estimates and taxonomy) is possible, and vice versa.

6 Experimental Results

The experiments were conducted with the prototype implementation of the algorithm. The system is realized C and Prolog and run on Sun Ultra-1 work stations. The results of the search runs are summarized in table 1. No_sc is the number of patterns generated by the specialization operator, sc is the number of patterns remaining for evaluation after complete pruning, t-sc is the time in seconds required for checking the subset condition, t-total is the total run time of the search run in seconds. Par is the parameter setting (σ_a for assoc, σ_m for midos, α for impint) of the search run. All search runs except the one marked with $*$ have completed the search. The search run marked with $*$ was stopped after about 22.5 hours. Until then, 77849 patterns had been evaluated.

Complete Pruning with Subset Condition versus Basic Specialization Operator. Figure 1 illustrates the gain of pruning with the subset condition versus basic pruning based on a specialization operator. The number of patterns that have to be evaluated is depicted on the y-axis, different parameter values are depicted on the x-axis. The graphs show that the effect of complete pruning is quite reliable, i. e., it occurs in all search settings. It reduces the number of patterns that have to be evaluated by about 25 to 50%. The columns (t-total) and (t-sc) in table 1 make clear that the cost of checking the subset condition is negligible compared to the total run time.

Pruning Based on Taxonomies. Figure 2 compares the numbers of evaluated patterns in the tax variant of the search space to the numbers of evaluated patterns in the no_tax variant. The graphs illustrate that the effect of pruning based on the taxonomies is not uniform over the different search runs. The result of the search run with $\sigma_m = 0.1$ with the KRK database in the midos setting is remarkable. Although the taxonomy based pruning clearly reduces the number of patterns, the search run takes longer (see table 1). The reason probably is that the taxonomy affects the order in which patterns are evaluated. The acceptance criterion of the midos k best search is dynamical; as the search progresses, better and better patterns are discovered, and the acceptance criterion becomes stricter.

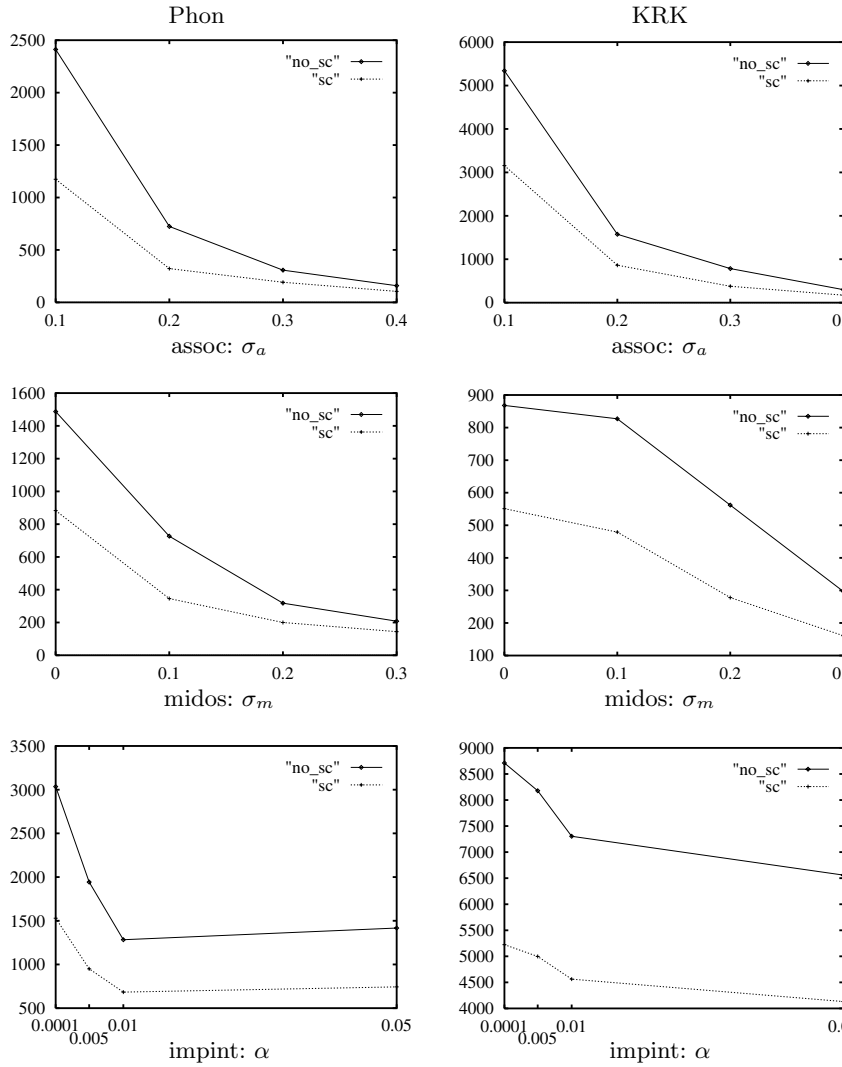


Fig. 1. Subset condition (sc) versus not using the subset condition (no_sc).

In the no_tax search run, some patterns that are expensive to evaluate can be pruned, whereas, in the tax search space, they have to be evaluated earlier when the acceptance criterion is still too weak to prune these patterns.

Effect of Optimum Estimates. The graphs in figure 3 illustrate the contribution of optimum estimates to pruning of the search space. In the midos setting, the graph labeled oe refers to search runs using optimum estimates du_{oe1} and du_{oe2}. The graph labeled weak_oe refers to search runs that use the optimum estimates du_{oe2} and the weaker version of du_{oe1}, du_{oe3}. The graph labeled no_oe refers to search runs that employ du_{oe2} and neither du_{oe1} nor du_{oe3}.

Table 1. Results of the search runs.

		Phon				KRK			
		#patterns		time (secs.)		#patterns		time (secs.)	
	par.	no_sc	sc	t-sc	t-total	no_sc	sc	t-sc	t-total
tax, opt_est, restrict_general									
assoc	0.1000	2412	1174	1	1173	5340	3157	7	2154
	0.2000	724	322	0	333	1575	862	0	700
	0.3000	307	192	0	199	784	378	0	333
	0.4000	159	105	0	117	300	173	0	169
midos	0.0001	1487	883	0	857	868	551	0	471
	0.1000	727	346	0	358	827	479	0	416
	0.2000	318	200	0	206	562	278	0	266
	0.3000	208	144	0	155	297	161	0	160
impint	0.0001	3035	1529	2	1452	8710	5224	23	3312
	0.0050	1943	950	0	923	8178	4998	26	3333
	0.0100	1283	684	0	696	7304	4562	18	3398
	0.0500	1417	744	2	769	6559	4132	13	3049
no_tax, opt_est, restrict_general									
assoc	0.1000	4543	1426	2	1375	7485	3421	9	2257
	0.2000	1577	424	1	408	2440	913	2	732
	0.3000	782	243	0	232	1355	465	0	381
	0.4000	428	171	0	160	525	203	0	188
midos	0.0001	5018	1805	1	1428	1725	629	0	433
	0.1000	1871	513	0	480	1475	573	0	414
	0.2000	1061	303	0	280	1090	380	0	340
	0.3000	631	209	0	197	865	314	0	276
impint	0.0001	11512	3281	4	2795	11890	5401	24	3382
	0.0050	8068	1962	2	1734	11000	5114	21	3363
	0.0100	6634	1672	2	1467	9645	4636	17	3251
	0.0500	5376	1573	2	1414	8700	4231	15	3182
tax, opt_est_weak, restrict_general									
midos	0.0001	-	77849	-	81000	41587	37254	1186	17728
	0.1000	1583	839	0	856	5959	3565	9	2639
	0.2000	454	275	0	291	2022	906	1	782
	0.3000	216	148	0	161	1009	571	1	504
impint	0.0001	13184	7091	27	6646	10774	7525	45	4427
	0.0050	8132	4008	10	3748	9406	6627	35	4194
	0.0100	5796	2963	6	2834	8170	5759	27	4058
	0.0500	5721	2932	5	2807	7269	5124	22	3659
tax opt_est no_restrict_general									
impint	0.0001	35048	19470	195	18661	29719	23744	454	12611
	0.0050	42900	24330	318	23630	30473	24391	486	12768
	0.0100	44478	25430	348	24673	30607	24466	496	12834
	0.0500	48737	28638	447	30660	31041	24865	510	13001

In the impint setting, the graph labeled oe refers to search runs using optimum estimate dII_{oe2}. The graph labeled weak_oe refers to search runs that use the weaker optimum estimate II_{oe1}. The graph labeled no_oe refers to search runs that do not use an optimum estimate for pruning.

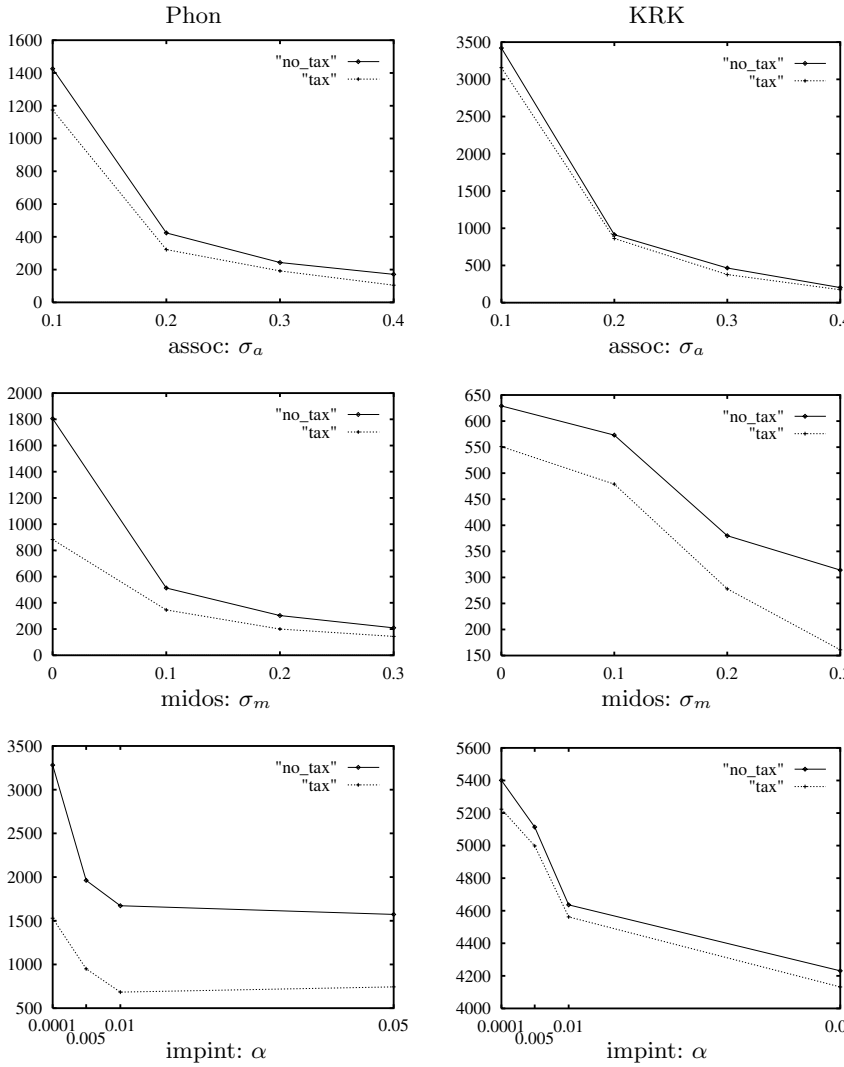


Fig. 2. Taxonomy (tax) versus no taxonomy (no_tax).

The figure shows that optimum estimates are a powerful and reliable means for pruning the search space. Furthermore, the impint search runs demonstrate that a minor improvement of an optimum estimate can have a great effect on its power.

Restriction to Most General Patterns. Figure 4 shows the number of evaluated patterns when the restriction to most general patterns applies (rgen) versus the number of evaluated patterns when the restriction is waived (no_rgen). The restriction leads to a considerable reduction of the size of the search space. In the

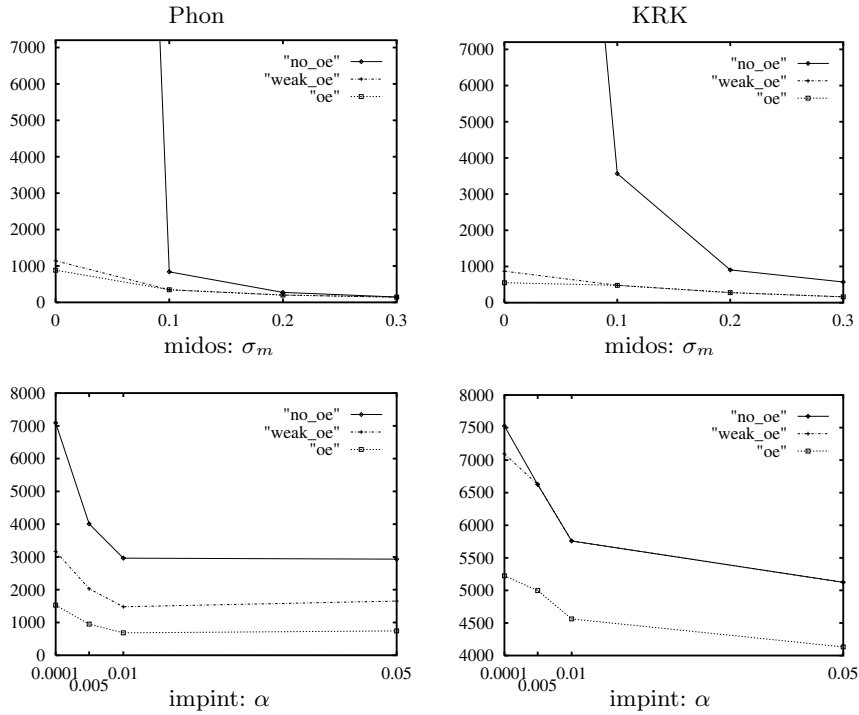


Fig. 3. Best optimum estimates (oe) versus weak optimum estimate (weak_oe) versus no optimum estimate(no_oe).

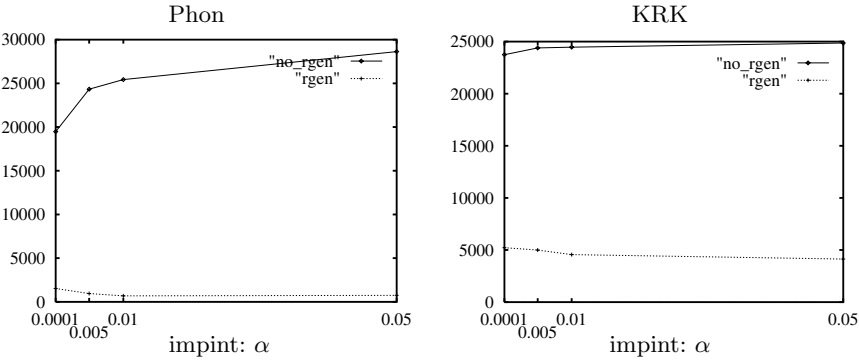


Fig. 4. Restriction on most general patterns (rgen) versus no restriction (no_rgen).

phonetics experiments, the number of accepted patterns ranged from 15 to 19 with rgen, and from 4010 to 11234 without rgen. In the KRK experiments, 6 to 9 patterns were accepted with rgen, and 10595 to 13590 patterns were accepted without rgen.

7 Conclusion

The experiments reported in the paper provide a comparison of different methods for pruning the search space for subgroup discovery in an ILP framework. In particular, optimum estimates and the subset condition have produced good and reliable pruning effects. The experiments have shown that pruning based on the subset condition can have a similarly good effect for search in a multi-relational data base as it has for search in a single-relation database. Instead of the subset condition, ILP systems typically use θ subsumption to detect subsumption relationships between patterns. However, computation of θ subsumption is very costly. Some existing ILP systems, therefore, only use what here is termed basic pruning for restricting the search space. The results of this paper indicate that it is worthwhile to design the pattern language or search space such that additional subsumption relationships can be exploited for pruning.

The paper intends to contribute to the development or improvement of algorithms for multi-relational subgroup discovery. Although the diversity among the results of the search runs indicates that the experiments do cover a reasonable range of application types, further experiments with different applications will be interesting. The design of artificial databases that systematically vary relevant characteristics of applications seemed not appropriate for experiments in this early stage of the research since the first-order setting offers so many possibilities for variation.

References

AMS+96. R. Agrawal, H. Mannila, R. Srikant, H Toivonen, and I. Verkamo. Fast discovery of association rules. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*. MIT Press, Cambridge, MA, 1996.

Deh98. Luc Dehaspe. *Frequent pattern discovery in first-order logic*. PhD thesis, K.U.Leuven, December 1998.

FDPB95. L. Fleury, C. Djeraba, J. Philippe, and H. Briand. Contribution of the implication intensity in rules evaluations for knowledge discovery in databases. In Y. Kodratoff, G. Nakhaeizadeh, and C. Taylor, editors, *Workshop Notes of the ECML-95 Workshop Statistics, Machine Learning and Knowledge Discovery in Databases*, 1995.

FL01. P. Flach and N. Lachiche. Confirmation-guided discovery of first-order rules with Tertius. *Machine Learning*, 42(1/2):61–95, 2001.

GL93. R. Gras and A. Larher. L'implication statistique, une nouvelle méthode d'analyse de données. *Mathématique, Informatique et Sciences Humaines*, (120), 1993.

Rap98. S. Rapp. Automatic labeling of German prosody. In *Proc. of Int. Conference on Spoken Language Processing (ICSLP'98)*, 1998.

Web00. I. Weber. Level-wise search and pruning strategies for first-order hypothesis spaces. *Journal of Intelligent Information Systems*, 14(2–3):217–239, 2000.

Wro97. Stefan Wrobel. An algorithm for multi-relational discovery of subgroups. In J. Komorowski and J. Zytkow, editors, *Proc. First European Symposium on Principles of Knowledge Discovery and Data Mining*. Springer, 1997.

Noise-Resistant Incremental Relational Learning Using Possible Worlds

James Westendorp

The University of New South Wales, Sydney, Australia
jhw@cse.unsw.edu.au

Abstract. Incremental learning from noisy data is a difficult task and has received very little attention in the field of Inductive Logic Programming. This paper outlines an approach to noisy incremental learning based on a possible worlds model and its implementation in NILE. Several issues relating to the use of this model are addressed. Empirical results are shown for an existing batch domain and also for an interactive learning task.

1 Motivation

Most research in Inductive Logic Programming (ILP) [8] has focussed on learning from batch data. However, there are many domains where the data is not available beforehand, and it may be expensive to obtain. Examples of this type of domain include the personalisation of user interfaces and learning user profiles or preferences such as email filtering rules [2]. In these cases, an incremental learner is more suitable.

Langley [7] describes a set of necessary attributes for an incremental learner to operate in these environments:

1. The learner must have no upper limit on how many examples it can process.
2. Each example must be processed in a reasonable amount of time.
3. Memory requirements must increase as a tractable function of experience.

Schlimmer & Fisher describe a similar set of properties in [16]. They refer to the need for rapid update of the knowledge base as each new instance is encountered. This corresponds to attributes 1 and 2 above. Item 3 above is quite important for learners in embedded systems. In these cases the resources available to the learner may be quite limited. For a learner to operate in these domains its memory usage must have a finite upper limit.

Two further requirements must be added for the learner to useful in a real-world setting:

4. The learner should be able to operate without an oracle.
5. The learner must be robust to the presence of noise in its training data.

The restriction on the use of an oracle reduces the reliance of the learner on the environment. Though useful, dependence on an oracle restricts the applicability of a learner to those domains where one can be provided.

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 317–332, 2003.

The ability to handle noise is a critical characteristic, but the limitations of incremental learning make it hard to manage. Specifically, the restriction of having to handle only a single example at a time makes it very difficult to know if any given example should be considered noise.

We will focus on the ability to handle noisy data, an issue that most incremental ILP learners have not dealt with. Specifically, we want to handle classification noise in examples.

This paper outlines an attempt to develop an incremental learner that satisfies these attributes within the ILP framework. The expressive power of Horn clauses makes them a more desirable representation than propositional logic. The drawback is that the expressiveness of the language makes for a far greater search space, so care must be taken to ensure that the learner still operates in a reasonable time frame.

2 Approaches to Noise Handling

There are several approaches that are used to handle noisy data when learning either propositional or relational rules. Most of these come from the batch learning framework, so their applicability to an incremental setting must be evaluated.

Pre-pruning methods [5] limit the size that a clause will grow while it is being created, by removing the requirement of perfect discrimination between positive and negative examples. Foil [13] is a batch learner that uses encoding length to determine when to stop growing a clause. Fossil [4] and mFoil [3] are Foil variants that use statistical techniques to determine when to stop clause growth. These approaches may be useful in avoiding over-fitting in an incremental setting as they could be made to operate without a complete training set.

We can rule out methods that require full knowledge of the dataset to operate, as they will not work within an incremental setting. This excludes noise-filtering pre-processing steps as well as post-pruning methods that require a pruning set. We can also rule out probabilistic batch methods that require advance knowledge of the entire training set.

Hydra [1] generates concept descriptions for all classes and uses a probabilistic approach when classifying unseen instances. This approach may have some benefit, as an incremental learner could also learn descriptions of all classifications. The downside would be the extra computational cost associated with learning both classifications.

There are some noise-resistant propositional learners such as Stagger [15] and Yails [19]. These apply weights to attribute values or ranges, and then use probabilistic methods to determine classification. This approach is not useful in a relational setting as there is no direct mapping from propositions to relations.

Hillary [6] is a relational rule learner that handles classification noise in examples. It achieves this by using a possible worlds approach, where each world contains a theory and some scoring information. When an example is presented that is inconsistent with the current theory, Hillary generates multiple new worlds. One of these is made on the assumption that the example is noise. In this case the example is rejected, the theory remains intact and the scores for that world are

updated accordingly. Hillary also attempts to incorporate the new example into the existing theory, which may generate several new possible worlds. When all worlds are generated, the best one is selected using a heuristic scoring function, and the rest are rejected. The scoring function is based on a tradeoff between the coverage of a world and its syntactic size.

Hillary learns theories by finding clauses that are the most specific generalisations of examples. Each clause in a theory has a small set of examples that it covers. These examples are the supporting evidence for that clause and guide the revision process. These sets are simple rolling windows, with the oldest examples being removed first once the size limit is reached.

A major limitation is that each example is assumed to contain sufficient information to enable the generation of a suitable generalisation, without the use of any background knowledge. The use of *relative-least-general-generalisations* [11] could alleviate this problem, although RLGG's can become very large. Additionally, the use of rolling windows for evidence means the evidence that caused a revision will eventually be lost and may be replaced by discriminating useful examples.

An approach based on the use of possible worlds seems the most useful in an incremental relational setting. It provides a simple way to address the unknown validity of an inconsistent example. Such a learner can generate worlds for both possibilities, then select between them.

The drawback is the increased processing required. Optimising a possible worlds learner to minimise the amount of extra work done is beyond the scope of this paper. It is an issue that would need to be addressed in order to develop learners that are useful in a real-world setting.

3 Approaches to Incremental Relational Rule Learning

A possible worlds approach does not specify the way revisions are done in order to make a theory consistent with a new example. A brief review of other existing incremental relational rule learners is undertaken in order to find an appropriate type of revision for a possible worlds model.

Shapiro's Model Inference System (MIS) [17] does general-to-specific learning of multiple concepts and guarantees that it can find a description in the limit. It requires noise-free data and an oracle. Both Cigol [10] and Marvin [14] are specific-to-general learners, using inverse resolution and absorption respectively. These systems also require an oracle and perfect data. Minerva [18] is a highly expressive learner that is interruptible and capable of anytime learning. Like MARVIN, it is does active experimentation in its environment, in order to obtain new facts.

In each of these cases, there is a requirement for an oracle of some sort, either explicitly or implicitly as part of the environment. As we have already mentioned, an oracle is useful, but may not always be available, so we would prefer an approach that doesn't require one. An approach is needed that can revise clauses based entirely on internal information, such as pre-pruning.

It is worth noting that most of these systems have been around for some time. The newest was developed in 1996. This reflects of the lack of work undertaken in incremental ILP in recent times.

3.1 Selecting a Revision Method

Both general-to-specific and specific-to-general searches fit a possible worlds framework. Both are also suitable for pre-pruning, so neither of these criteria is sufficient to determine an approach. We look instead at the usefulness of the searches in a general incremental setting with potentially noisy data. It is assumed that clauses are independent of each other and that there is a small set of examples to guide the search.

In a specific-to-general search, an example is used as the most specific starting point, and the search generalises from there. This is a problem if the example is incorrect, as the set of all clauses that are more general than it will not contain the target theory. A general-to-specific search, on the other hand, has at least a possibility of staying on the right track to the correct theory when presented with a noisy example, as there may be a satisfactory clause that is more general than both the faulty example and the target theory.

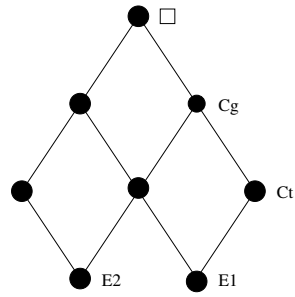


Fig. 1. A simple refinement graph

This is demonstrated in Fig 1. E_1 is a correct positive example, while E_2 is a negative mislabelled as positive. C_t is the target clause. There may exist a clause C_g that is more general than both E_2 and C_t, and is therefore a potential clause that covers E_2. In a specific-to-general search, we cannot reach C_t directly. Even if the faulty example is removed from the evidence set at a later date and other covered examples are added, we would still need to re-specialise. In a general-to-specific search, however, we may find C_g is sufficiently specific as not to warrant further specialisation. From here, if the offending example is removed at a later date, further re-specialisation may lead directly to the theory.

A second reason for choosing a general-to-specific approach is that it is biased towards syntactically smaller clauses. This has two advantages. Firstly, it makes the generated theories easier to understand and work with and secondly, the clauses are smaller to store.

```
function NILE

initialise w, the current world
loop (forever)
    e = next example;
    Update information from e
    for each fact f in e
        W = PROCESSEXAMPLE (w,e)
        Remove agreed rejections from all worlds in W
        w = best world in W
    end for
end loop
```

Fig. 2. The main learning algorithm for Nile

4 Implementation

The previous sections have enabled us to establish a general approach to noise handling and the type of revision that will take place. We now describe the design in more detail as it is implemented in Nile.

4.1 General Information

Nile is a single concept learner, although its implementation allows for several learners to be batched together, allowing multiple concepts to be learned concurrently. Nile is not a true multi-concept learner as the different concepts are maintained and revised independently of each other. The description of the system that follows applies to the learning of a single concept.

The concept language is non-recursive, function-free Horn clauses. The non-recursion limitation is a result of Nile's evidence storage approach. Positive evidence is stored on a clause-by-clause basis and each clause is treated independently, whereas recursive theories require dependence between clauses. The function-free restriction is done primarily for simplicity, and is not a large restriction since clauses containing functions can be flattened [12].

4.2 Nile's Learning Algorithm

Nile begins with a single world containing the empty theory. It also starts with background knowledge applicable to all examples. This background may be intensional or extensional. Any metalanguage must also be given to Nile on startup. Pseudocode for the main algorithm for Nile is shown in Fig 2.

When Nile receives an example it is pre-processed to extract any new predicates and constants that may be present. These are added to the current language. An example consists of one or more positive or negative ground facts, and possibly some additional background relating to those facts. The additional background is retained only for as long as the examples that it is related to

```
function ProcessExample (World w, Example e)

W = Learn(w, e)
for each w' in W
    if w' contains unlearned examples
        u = the first unlearned example in w'
        remove u from the unlearned list for w'
        W' = Learn(w, e)
        W = W + W' − w'
    end if
end for
return W

function Learn (World w, Example e)

intialise W =, a set of possible worlds
if e is consistent with w
    w' = w + e as supporting evidence
    add w' to W
else if e is positive
    W = W+ AddGeneralise (w, e)
    W = W+ GeneraliseSpecialise (w, e)
    W = W+ Reject (w, e)
else
    W = W+ SpecialiseOrRemove (w, e)
    W = W+ Reject(w, e)
end if
for each w' in W
    CheckSubsumption(w')
    CheckCoverage(w')
end if
return W
```

Fig. 3. The learning algorithm for a single world in Nile

are still in use, minimising space usage. If an example contains more than one positive or negative fact, each fact is processed separately.

Each new fact is presented to the current world in turn. This process generates a new set of possible worlds that are then post-processed. Finally, the best is selected and the rest rejected.

4.3 Example Processing within a Single World

When an example is presented to a specific world, Nile first checks if it is consistent with the current theory. If a positive example is consistent, it is added as supporting evidence to the clause that covered it. If a negative example is consistent, it is added to the negative evidence stored for the entire theory.

If an example is inconsistent Nile applies its positive or negative revision operators to the original world to generate a new set of worlds. These operators may each generate new worlds. A subsumption check is then performed on each new world to ensure that no clause in the theory is subsumed by another clause. If a subsumed clause is found, it is removed and its evidence is passed to the subsuming clause.

It is possible that the application of a revision operator, either for the current step or at some previous time, will have removed a faulty clause. When this occurs, the positive evidence for the removed clause needs to be re-learned. This process can become quite costly, particularly since re-learning may take place for several worlds. This cost is reduced by only allowing one example to be re-learned at each step. The rest can be re-learned at a later date if the world is retained. The re-learning process is exactly the same as for the original learning process.

The revision operators may also considering an example and so reject it. However, the example may have been correct and may become consistent with the world at a later date, so we do not want to discard it immediately. Each world contains a list of examples that have been rejected. After learning has taken place, any of the rejected examples or the examples awaiting re-learning that has now become consistent with the current theory is placed in the appropriate evidence set.

4.4 The Revision Operators for a Single World

There are four revision operators in Nile. Each operator make some changes to the world, either to its theory or its additional information, then updates the worlds scores accordingly. The world's scores are the number of positive and negative examples covered or uncovered, and the number of steps that the its theory has been constant. These values are used to determine a world's strength.

AddSpecialise creates a new empty clause to cover a positive example. It then specialises the clause in order to exclude the world's negative evidence. If a suitable clause can be found, it is added to the world with the new example as supporting evidence. It is possible that no suitable clause will be found in this way.

GeneraliseSpecialise attempts to generalise an existing clause to cover a positive example. For each clause in the theory, it generalises until the incoming example is covered. It then re-specialises the generalised clause in order to exclude any of the world's negative evidence that has become covered by the generalisation process. As with the Add operator, it is possible that no suitable revision will be found.

SpecialiseRemove operates on all clauses that incorrectly cover a negative example. There are three steps in this process for each clause. Firstly, the operator checks if any of the supporting evidence for the faulty clause is covered by other clauses. If this occurs, then the evidence is moved to the other clause that covered it. It then attempts to find a specialisation of the faulty clause that excludes the new negative example while still covering the remaining evidence. If

a suitable specialisation is found, the revised clause replaces the faulty clause. If no specialisation can be found, the faulty clause is removed, and its supporting evidence needs to be re-learned.

The REJECT operator applies to both positive and negative examples. It considers the example to be noise, and makes no changes to the world's theory. The example is added to the set of rejected examples for the world.

Within each of the three revising operators, Nile must search for a suitable revision, either a specialisation or generalisation of an existing clause. Each operator uses the same search process to do this.

The search uses MIS-style specialisation steps (add predicates, bind variables and substitute constants) or their equivalent generalisations. It has several constraints on it in order to manage the search space. Firstly, a beam search is used to reduce the chance of local maxima without the unmanageable cost of doing a complete search. Beam selection is based on a clause's coverage, or syntactic size if coverage is equal. Secondly, limits are placed on the size of the clause body and the number of free variables allowed at any stage of the revision search. The default values for these are three and two respectively. Thirdly, Nile uses metalanguage information if available, incorporating functional modes, types and symmetry. If mode information is not available, any new predicate added to a clause during revision must contain a variable already in the clause.

4.5 Handling Noise in a Possible World

A potential problem with the design thus far is the possibility of examples in the evidence sets being incorrect. This can cause generation of clauses that are over-fitted to specific incorrect evidence. The heuristic nature of the possible world selection process means that these worlds may still be selected, so the issue needs to be addressed. An added problem is that the limited amount of stored data makes it difficult to determine which evidence is faulty.

Firstly, In order to reduce the chance of clauses being over-fitted, Nile specifies a constant E_{min}, the minimum number of positive examples that must be covered by a revision for it to be allowed. By setting $E_{min} > 1$, it becomes more difficult for a noisy example to be covered, since a clause must cover other examples while still discriminating against the negative examples. A small change is therefore necessary for ADDSPECIALISE operator to still work, as it only processes a single example at a time. We allow it to use as additional evidence any rejected positives or any positive evidence awaiting re-learning. A valid revision for ADDSPECIALISE must cover the incoming example and $E_{min} - 1$ of the unlearned or rejected examples.

Secondly, Nile relaxes the constraints on the search for revisions. It allows a revision to be inconsistent with a small number of evidence examples and still be considered valid. In this way an incorrect example in an evidence set may become inconsistent with the resulting revision, but a more suitable revision may be found. By default, Nile allows one positive and two negative to be covered, provided at least E_{min} other positive examples are covered. If this occurs then the inconsistent evidence is now considered to be noisy. If positive evidence is

inconsistent, it is removed and placed in the rejected list for the given world. If negative evidence is inconsistent, it is not removed immediately, but is tagged as being inconsistent with the resulting revision.

Recovering from over-fitting also requires a bi-directional search, so that revisions can be made that reverse prior over-fittings. This happens explicitly in the GeneraliseSpecialise operator, and implicitly as a result of removing and then re-learning a clause.

Evidence Example Selection. Nile stores examples as evidence to help guide the search for future revisions, but we have not as yet determined which examples to store. As the stored evidence is the primary means by which revisions are to be selected, it is important to select useful examples for revision. Specifically, we want to store the most discriminating examples we can, so that any future revisions are less likely to undo the current revision.

Nile has two methods for determining the examples that are most significant.

For a positive example, it is determined by whether or not an example caused a clause to be created or revised. If this is the case the evidence is marked to distinguish it from evidence that was merely covered by the clause.

A negative example is considered significant if it caused a clause to be specialised. It is also considered significant if it was the last negative clause to become uncovered during an AddSpecialise or GeneraliseSpecialise revision. As a negative example may cause the revision of more than one clause, each example negative evidence has an attached list of the clauses it caused to be revised, and a counter of the total number of revisions it has caused.

When more than the maximum number of positive examples are present as evidence for a clause, preference is given to retaining examples that caused a revision. For negative examples, preference is given to those that have caused more revisions and are covered by fewer clauses. Where two negative evidence examples have the same score, preference is given to the one that has caused the most revisions. Excess evidence removal is the last step in the world revision process. By default, Nile stores 5 positives per clause and 20 negatives for a theory.

4.6 Post-processing

So far no mention has been made of when a world's rejected evidence is removed, although it must be removed at some stage. A buildup of rejected examples violates the bounded memory constraint for incremental learning. Furthermore, if a rejected example is incorrect, we do not want to have to keep checking it or have it become support for an incorrect clause.

Nile removes rejected examples as part of the post-processing phase of world generation. Nile considers an example to be noise if it is rejected in all the new possible worlds, and each possible world has rejected it for more than a specified minimum number of steps. By default this value is 10. If one of the worlds is an empty theory or contains a clause with no body, then this step is skipped, in order to avoid rejecting all positive or all negative examples.

Nile now has to determine which world to keep. The question of which world to keep and which ones to throw away is critical. A heuristic function is needed to give us an idea of each worlds strength. There is a danger when using heuristics of being caught in local minima, however this is alleviated somewhat by Nile's ability to do bi-directional search. The primary factors to consider are the world's accuracy and size, but also its stability. We would like the learner's classification strategy to be reasonably constant over time, rather than changing with every example seen. Each of these factors is addressed in Nile's scoring heuristic and world selection strategy.

The World Score Heuristic. We use a heuristic that extends Hillary's scoring metric. Hillary was designed to explicitly trade-off simplicity and coverage, and used the following heuristic:

$$\begin{aligned} score(W) &= \omega * coverage(W) + (1-\omega) * simplicity(W) \\ coverage(W) &= \frac{C_{pos} + C_{neg}}{C_{pos} + R_{pos} + C_{neg} + R_{neg}} \\ simplicity(W) &= any\ mapping\ of\ size\ \rightarrow [0:1] \end{aligned}$$

Here C_p and C_n are estimates of the number of correct positives and negatives in W, while R_p and R_n are estimates of the number of rejected positives and negatives in W. The parameter ω controls the tradeoff between coverage and simplicity. A major drawback with this scoring function is that it does not take into account the proportions of positive and negative examples seen, and as such can favour worlds that stay consistent by rejecting a disproportionate number of positives or negatives. Further, it is lacking a measure of stability.

The scoring function in Nile extends that of Hillary to cover both these areas, as shown below:

$$\begin{aligned} score(W) &= \omega * coverage(W) + (1-\omega) * simplicity(W) + stability(W) \\ coverage(W) &= \frac{1}{2} * \frac{C_{pos}}{(C_{pos} + R_{pos})} + \frac{C_{neg}}{(C_{neg} + R_{neg})} \\ simplicity(W) &= \frac{K_{sim}}{K_{sim} + syntacticsize(W)} \\ stability(W) &= min(s_c * nsc(W), s_{max}) \end{aligned}$$

Better proportioning of rejected examples is handled by the coverage function. It favours worlds where the proportion of rejected positive to negative examples is closer to the proportion of total positives and negatives seen. The syntactic size of a theory is simply the sum of the number of clauses, predicates, constants and variable bindings in it. The simplicity equation uses an inverse function rather than a linear function so that it asymptotes towards zero in the limit rather than becoming negative. The default value for K_{simp} is 20.0, chosen because it provides a fairly slow rate of decrease. There is a degree of flexibility with the value here, as the simplicity factor normally has quite a low impact on the overall score.

Score: C_{pos}: 2 C_{neg}: 6 R_{pos}: 2 R_{neg}: 1	
Num Steps Constant: 2	
Clause 1:	**gp(V0,V1) ← p(V0,V2),p(V2,V1).**
Evidence:	*gp(harry,lisa).
	gp(robyn,tanya).
Negative	~gp(robyn,jacob). `rev: [1] cov: [] tot: 1`
Evidence:	~gp(bill,emma). `rev: [1] cov: [] tot: 1`
	~gp(hendrik,gonni). `rev: [] cov: [1] tot: 0`
Rejected Positives:	gp(betsy,anthony) `(3)`
Rejected Negatives:	
Re-learn Positives:	gp(tony,fred)

Fig. 4. An example possible world in Nile

S_c is the constant scale factor for stability of a world, and S_{max} is the maximum allowable stability value for a world. Currently the defaults are 0.005 and 0.05 respectively. These provide a simple way to slightly bias towards more stable worlds without overly affecting the other parts of the score. $nsc(W)$ is the number of steps for which the theory in world W has been constant.

4.7 Putting It All together

An example of a possible world is shown in Fig 4, showing the different parts of each world. In the example, we have a world with one clause in its theory, shown in bold. This clause has two examples as supporting evidence for it. The first, as revising evidence, is marked with a '*'. There are three negative examples that provide negative evidence for the entire theory. The first two negative evidence examples caused revisions in clause 1, while the third negative evidence example is covered by clause 1. The world contains one rejected positive that has been rejected for three steps, and one positive example awaiting re-learning.

5 Results

We have tested Nile in two domains. Firstly in an existing domain with well known properties, and then in an incremental domain.

5.1 Learning Illegal Chess Positions

KRK-illegal [3,4] is a well-studied domain, and the knowledge of its properties gives us a greater ability to analyse the results. We use Progol [9] in rolling-window form to provide a baseline. The window size is set to 100 and Progol is re-run after the current theory has misclassified five examples.

Our only change to the standard KRK dataset is to add Row and Column type information in order to restrict the search. We add the types and intermediate predicates that use this type information, such as $rowadj(A, B) \leftarrow adj(A, B)$.

Tests were run for several noise levels, where the noise level is the chance of an example's class label being switched before presentation to the learner. For

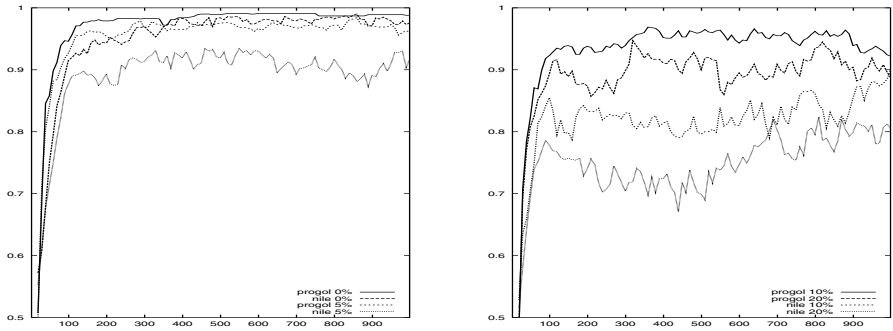


Fig. 5. Average Accuracy for KRK domain

	Running time(sec)				Num examples stored			
	0%	5%	10%	20%	0%	5%	10%	20%
Progol	37	137	243	438	100	100	100	100
Nile	46	157	288	410	60	76	81	85

Fig. 6. Running times and number of examples stored for KRK domain

each level we ran 20 trials, each consisting of 1000 examples. Every 10 examples, the current theory was run over an independent test set of 100 examples. These test results were then averaged across the trials. The results are shown in Fig 5. Average trial running time is shown in Fig 6, as is the number of examples stored. This value is the total number of examples within the current possible world, including ignored evidence and evidence awaiting re-learning.

Even with no noise in the data, Nile is not quite able to reach 100% accuracy. This is evidence of the heuristic nature of Nile. Nile is unlikely to learn complex clauses that cover very few examples, as it interprets their infrequency as noise. For instance, it never learned the case where the white king is between the white rook and the black king.

Progol was consistently more accurate than Nile, although Nile was still able to learn reasonably accurate theories, particularly at lower noise levels. This may be related to the concept being learned, as it is possible to get reasonable accuracy with only two or three simple clauses. Even with some noise, a fairly accurate target theory is still attainable. It is worth noting that most of the Nile's accuracy error was in classification of negative evidence, indicating that an over-general theory has been learned. This is partly a limitation of using smalls sets of evidence to guide the search, resulting in clauses that over-fit the current training data. It is also partly due to the use of a heuristic hill-climbing learner in an incremental setting. The negative examples that would cause the revision of an over-fitted clause may be rejected as noise if the resulting revision has a large heuristic cost than simply rejecting the example.

Progol and Nile had comparable running times, however Nile required fewer stored examples than Progol. This is significant as we want to limit the total

size used by the learner, although storing additional clauses may be required in order to gain increased accuracy.

5.2 Learning a Tetris-Style Game

The KRK domain gives us some insight into the system's behaviour, however we would like to apply to to an incremental domain, to see how it behaves when faced with a more complex problem. We would like a domain we can control, so we have created Tris, a scaled-down version of the popular blocks game Tetris. This allows us to mimic the same type of learner-environment interaction found in systems such as email sorters like *i-ems* [2]. Here the learner works by predicting a mailbox folder for a new message. The user then has the option of placing the message in a different folder. We use the same control approach in Tris, with the learner predicting a location for a piece that the game controller (either human or automated) can then override. If the prediction is satisfactory, the learner is given a single example that will be consistent with its theory. If the prediction was not satisfactory, the learner receives two training examples, representing the initial incorrect placement and the revised placement.

The are two major differences between Tris and Tetris. Firstly Tris has no real-time element, as we are not interested in anytime learning. Secondly, Tetris pieces consist of four squares, while Tris pieces only have three. This reduces the number of piece / orientation combinations (such as north-south line) from 19 to six: two orientations for a line piece and four for a corner piece. Background predicates for each step describe the current state of the board, namely the height of each column and whether a column is the lowest or highest column. Additional global background for the entire training run includes the *nextrow*, *notnextrow*, *nextcol*, *rownotequal* and *colnotequal* predicates.

The learner's aim is to learn a concept description that describes when a piece should not be placed in a particular column with a particular orientation. For example, *dont_place_corner_se(C)*←*nextcol(C,C2),colheight(C2,0)* . Hand-crafting a control theory revealed that the rules required to learn piece placement require exception predicates, outside the scope of Nile's ability. Instead, we learn where not to place pieces.

The 'player' for the experiments is a computer controller using the hand-crafted control rules. Its strategy is to ensure that wherever possible, no dropped piece will have empty space underneath it. Also, pieces are placed in an orientation that will let them drop as far as possible. Where multiple possible piece orientations are available, it favours those orientations that have occurred less often. If there are still multiple satisfactory placements available, it chooses randomly between them.

The only change made to Nile for these experiments was to increase the maximum clause body size to five. This was necessary to enable Nile to learn the longer clauses in the controller's target concept.

For each noise level, 20 trials were run and averaged. As Tris is a dynamic domain, we do not have a test set to use. Instead we measure accuracy as the the number of times that the learner made an acceptable piece placement. The

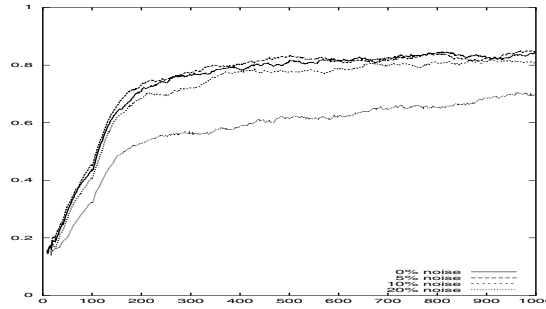


Fig. 7. Accuracy on training data for the Tris domain

results are the accuracy for the last 100 pieces seen by the learner at any point, and are shown in Fig 7.

With no noise in the system, Nile was not able to learn a complete description of the players strategy. It learned all the basic rules, but some of the rules are quite specific and don't occur very often. These cases are again dismissed as noise. Also, Nile's revision search is often sidetracked by irrelevant features that provide a more immediate discrimination. This is a susceptibility when using small evidence sets.

Unlike the KRK domain, the results for 5% and 10% are almost as high as for 0% noise. This shows again the limitations of what Nile is able to learn, but also that Nile is resistant to the presence of noise.

6 Conclusions

In this paper we have argued that there is a need for incremental learners that are resistant to noise. A survey of current noise-handling techniques showed a possible worlds model to be a useful approach, providing a simple way to handle uncertainty about the validity of examples. A design using this approach was presented, with attention paid to the issues of example selection and information required to handle noise. A theory revision process was outlined that would enable the system to run in a reasonable time. This approach has been implemented in Nile.

Empirical testing indicated that Nile is able to learn reasonably accurate theories in the presence of moderate levels of noise, with time and space constraints comparable to a rolling-window version of progol. These results showed the design's potential, but also revealed some limitations. The use of beam search in clause revision is beneficial as a way of making search times manageable but raises new problems. In conjunction with small evidence sets, the system becomes susceptible to over-fitting the current evidence. Clauses that require determinate literals are also less likely to be found. The use of heuristics means that the over-fitted clauses may not be removed even if conflicting evidence subsequently appears.

This is very much preliminary work, and there are several areas where more work needs to be done. The clause revision process needs to be improved to reduce the levels of over-fitting. A more formal model would also be desirable. There is also need for more testing in incremental domains with varied characteristics.

Acknowledgements

The author wishes to thank Mark Reid, Claude Sammut, Philip Rutgers and Tabatha Aylen. Without their assistance this paper could not have been written.

References

1. K. Ali and M. Pazzani. HYDRA: A noise-tolerant relational concept learning algorithm. In R. Bajscy, editor, *proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1064–1071. Morgan Kaufmann, 93.
2. E. Crawford, J. Kay, and E. McCreath. Automatic induction of rules for e-mail classification. In *Proc. of the sixth Aust. Document Computing Symposium*, 2001.
3. S. Dzeroski. Handling imperfect data in inductive logic programming. In *Proceedings of the fourth Scandinavian Conference on Artificial Intelligence*, pages 111–125. IOS Press, 1993.
4. J. Fürnkranz. Avoiding noise fitting in a FOIL-like learning algorithm. In F. Bergadano, L. De Raedt, S. Matwin, and S. Muggleton, editors, *Proc. of the IJCAI-93 Workshop on Inductive Logic Programming*. Morgan-Kaufmann, 1993.
5. J. Fürnkranz. Pruning algorithms for rule learning. *Machine Learning*, 27(2):139–171, 1997.
6. W. Iba, J. Wogulis, and P. Langley. Trading off simplicity and coverage in incremental concept learning. In *Proceedings of the fifth International Conference on Machine Learning*, pages 73–79, 1988.
7. P. Langley. Order effects in incremental learning. In P. Reimann and H. Spada, editors, *Learning in humans and machines: Towards an interdisciplinary learning science*. Elsevier, 1995.
8. S. Muggleton. Inductive logic programming. In S. Muggleton, editor, *Inductive Logic Programming*, pages 3–27. Academic Press, London, 1992.
9. S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13(3–4):245–286, 1995.
10. S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning*, pages 339–352. Morgan Kaufmann, 1988.
11. S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*. Ohmsha, 1990.
12. S.-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *LNAI*. Springer-Verlag, 1997.
13. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.
14. C. Sammut and R Banerji. Learning concepts by asking questions. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Vol 2.*, pages 167–192. Morgan Kaufmann, 1986.

15. J. Schlimmer. Incremental adjustment of representations for learning. In *Proc. of the fourth Int. Workshop on Machine Learning,*. Morgan Kaufmann, 1987.
16. J. Schlimmer and D. Fisher. A case study of incremental concept induction. In *Proceedings of the fifth National Conference on Artificial Intelligence*, pages 496–501. Morgan Kaufmann, 1986.
17. E. Shapiro. An algorithm that infers theories from facts. In A. Drinan, editor, *Proceedings of the seventh International Joint Conference on Artificial Intelligence*, pages 446–451, Los Altos, CA, 1981. Morgan Kaufmann.
18. K. Taylor. *Autonomous Learning by Incremental Induction and Revision*. PhD thesis, Australian National University, 1996.
19. L. Torgo. Controlled redundancy in incremental rule learning. In P. Bradzil, editor, *Proceedings of the European Conference on Machine Learning*, volume 667 of *Lecture Notes in AI*, pages 185–195, Berlin, 1993. Springer-Verlag.

Lattice-Search Runtime Distributions May Be Heavy-Tailed

Filip Železný[1], Ashwin Srinivasan[2], and David Page[3]

[1] Dept. of Cybernetics
Faculty of Electrical Engineering
Czech Technical University
Karlovo Nám. 13, 121 35 Prague, Czech Republic
zelezny@fel.cvut.cz
[2] Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford OX1 3QD, UK
ashwin@comlab.ox.ac.uk
[3] Dept. of Biostatistics and Medical Informatics and Dept. of Computer Science
University of Wisconsin
1300 University Ave., Rm 5795 Medical Sciences
Madison, WI 53706, USA
page@biostat.wisc.edu

Abstract. Recent empirical studies show that runtime distributions of backtrack procedures for solving hard combinatorial problems often have intriguing properties. Unlike standard distributions (such as the normal), such distributions decay slower than exponentially and have "heavy tails". Procedures characterized by heavy-tailed runtime distributions exhibit large variability in efficiency, but a very straightforward method called *rapid randomized restarts* has been designed to essentially improve their average performance. We show on two experimental domains that heavy-tailed phenomena can be observed in ILP, namely in the search for a clause in the subsumption lattice. We also reformulate the technique of randomized rapid restarts to make it applicable in ILP and show that it can reduce the average search-time.

1 Introduction

In the recent paper [4], Gomes et al. observe that procedures for solving propositional satisfiability problems exhibit a remarkable runtime variability. The runtimes vary greatly depending on the choice of a particular heuristic, a given problem instance, or - for stochastic methods - on the choice of different random seeds (initial truth assignments), or on another source of randomness. Often a satisfiability procedure "hangs" on a given problem instance, while a different stochastic run solves the same instance quickly. Even for a deterministic procedure and a given problem instance, small amount of randomization (e.g. in the employed heuristic) yields again largely varying search-costs, some of which are substantially lower than that of the deterministic algorithm.

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 333–345, 2003.

In their empirical study it is shown that distributions of the runtimes of many search algorithms decay slower than exponentially and asymptotically have *heavy-tails*. Unlike standard probability distributions, such as the normal distribution, where events that are several standard deviations from the mean are very rare, in heavy-tailed distributions there is a non-negligible probability that an event with an extremely high cost occurs. For example, in one of the studied problems, 80% of the runs solve the problem in 1,000 backtracks or less, however 5% of the runs do not result in a solution even after 1,000,000 backtracks. Gomes et al. believe that the heavy-tailedness is a property of many exhaustive backtrack algorithms for solving hard combinatorial problems, and offer a technique called *randomized rapid restarts* that exploits this property in order to reduce the average search-time. The technique was used to find solutions of previously unsolved instances of hard combinatorial problems.

Although many search problems give rise to a heavy-tailed distribution, others do not [5]. Our aim is to find out whether heavy-tailed runtime distributions occur in ILP. Namely, we empirically study the runtimes of the search for a first-order clause with defined desired properties, in the lattice imposed by the subsumption relation. Furthermore we reformulate the randomized rapid restarts algorithm to be applicable on the ILP search problem and on two important ILP benchmarks we evaluate whether it reduces the search-cost with respect to a deterministic exhaustive search.

The following section defines formally the notion of a heavy-tailed distribution and describes the method of randomized rapid restarts. Since the method requires randomization of the exhaustive search, Section 3 describes our way of randomization of the lattice search, based on the selection of a random starting clause (seed). The core of the study is Section 4 which will test empirically the hypothesis that heavy-tailed runtime distributions describe the clause lattice-search using benchmark ILP problems. In the same section, we shall also apply the technique of randomized rapid restarts on the same domains, and investigate whether it improves search efficiency. We summarize our observations in Section 6.

2 Heavy-Tailed Distributions and Randomized Rapid Restarts

The cumulative probability distribution $Pr(X < x)$ of a random variable X is a non-decreasing real function on the real interval $-\infty < x < \infty$ and will be denoted $F(x)$, i.e. $Pr(X > x) = 1 - F(x)$. Standard probability distributions have exponentially decreasing tails, e.g. for the standard normal distribution F_n it holds that

$$(1 - F_n(x)) \sim \frac{1}{x\sqrt{2\pi}} \exp \frac{-x^2}{2} \tag{1}$$

where $g(x) \sim h(x)$ denotes $\lim_{x\to\infty} g(x)/h(x) = 1$.

Recently, in the area of algorithms for hard combinatorial problems [4] but also other areas such as statistical physics, economics etc., different phenomena

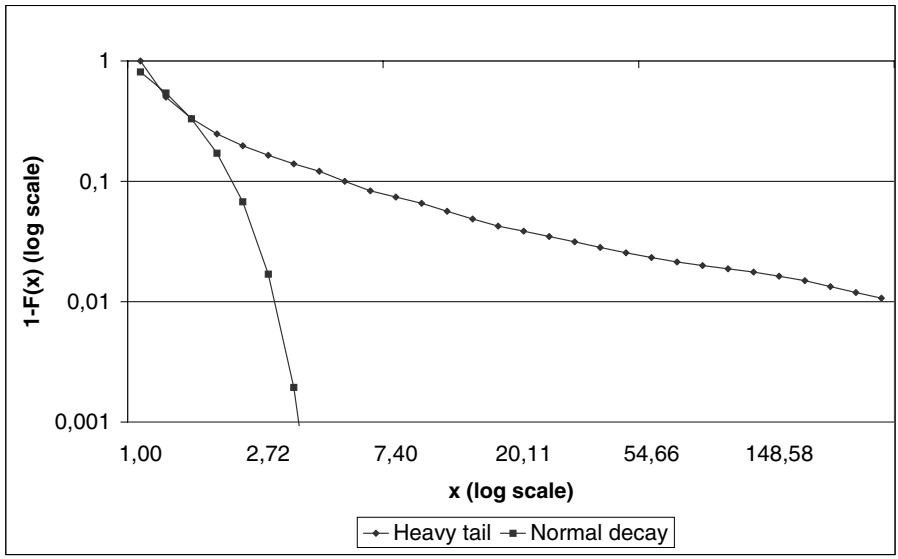


Fig. 1. Example of normal and heavy-tailed distributions on a log-log scale. The normal distribution decays faster than linearly while the heavy-tailed distribution decay shows an approximately linear decay.

have been shown to obey heavy-tailed distributions which often lead to non-intuitive behaviour. For this class of distribution it holds that

$$(1 - F(x)) \sim Cx^{-\alpha}, x > 0 \tag{2}$$

where $0 < \alpha < 2$ and $C > 0$ are constants. It is remarkable [4] that such distributions have finite mean but no finite variance if $1 < \alpha < 2$. If $\alpha \leq 1$, the distribution has even neither a finite mean nor a finite variance.

To determine whether a distribution estimated by a series of measurement has a heavy-tailed nature, i.e. it does not decay exponentially, we plot the measured distribution values in a diagram with both axis logarithmically scaled, because an exponentially decreasing distribution should show a faster-than-linear decay in the log-log scale. For example, substituting x with $\exp(x)$ in the normal distribution decay (see Eq. 1) and taking log yields

$$\log\{1 - F_n(\exp[x])\} \sim -\left(x + \frac{\exp(2x)}{2}\right) \tag{3}$$

while the same operation on the heavy-tailed distribution decay (see Eq. 2) yields $-\alpha x$, i.e. a heavy-tailed distribution should exhibit an approximately linear decay on the log-log scale as x approaches infinity.

Let us now consider an exhaustive-search algorithm randomized in such a way, that it starts the search at a randomly chosen point of the search space (seed). Depending on the particular type of a search problem and algorithm,

the distribution of times required to reach a solution from such seeds may or may not be heavy-tailed[1]. If a heavy tail is observed, the runtime variance and mean may be infinite and there is a non-negligible probability that a chosen seed will start an extremely costly search, although many other seeds may produce a quick path to the solution. A direct way to reduce the variance and mean in such a case is to run the exhaustive search up to a certain cutoff point and then restart at a different seed if a solution is not found. Clearly, this approach called *randomized rapid restarts* avoids the algorithm from getting trapped in a very costly path and exploits the high chance of obtaining an essentially shorter path in the next trial. It is shown in [4] that the randomized rapid restarts technique is superior to deterministic exhaustive searches in many propositional domains.

3 Randomizing the Lattice Search

We consider the normal ILP problem [10], namely we assume the sets of positive (negative) examples E^+ (E^-) and background knowledge B. We also assume that a search lattice of legal clauses has been defined by the generality (subsumption) relation, a clause mode language $\mathcal{L}$, and bounded by the most specific element $\perp$ (the bottom clause). This is a usual assumption of ILP systems based on the concept of inverse entailment [9], such as Aleph [6] and Progol [9].

The standard approach of conducting the lattice search is to start with the most general (most specific, $\perp$) clause and then proceed in a top-down (bottom-up) manner. However, starting the search with a different clause from the interior of the lattice may lead to a shorter runtime needed to reach the desired clause. Our plan is to investigate the distribution of such runtimes when the starting clauses are selected randomly from the lattice. If this distribution proves to be heavy-tailed, we will be able to utilize the technique of randomized rapid restarts to avoid the extreme-cost paths and improve the average performance.

The randomized search algorithm proceeds as follows. Some number of times (maxtries), the algorithm will carry out a short search (bounded by maxtime, Section 4). Each search begins by stochastic selection of a starting clause. The search is a deterministic best-first search, with heuristic function $h = pos(C) - neg(C)$. Here $pos(C)$ is the number of positive examples deducible from $C \wedge B$ [2], and $neg(C)$ is analogous for negative examples. From a given clause C, the neighbors of C in the search space are defined by a nontraditional refinement operator ρ. It differs from usual refinement operators employed in the top-down search of the mentioned systems in that it produces the set $REFS = \rho(C)$ of all neighbours of C in the lattice, i.e. also including clauses that subsume C. Therefore this kind of search can be seen as radial, rather than top-down or

[1] Gomes et al [5] discover the heavy-tailedness of different search problems purely empirically and report that further studies are needed to determine exactly what characteristics of combinatorial search problem lead to heavy-tailed behaviour. In this ILP-focused study we also take an empirical approach.

[2] In the case when C is constructed as an extension of an existing partial theory H in a greedy cover search, we assume that H has been added to B.

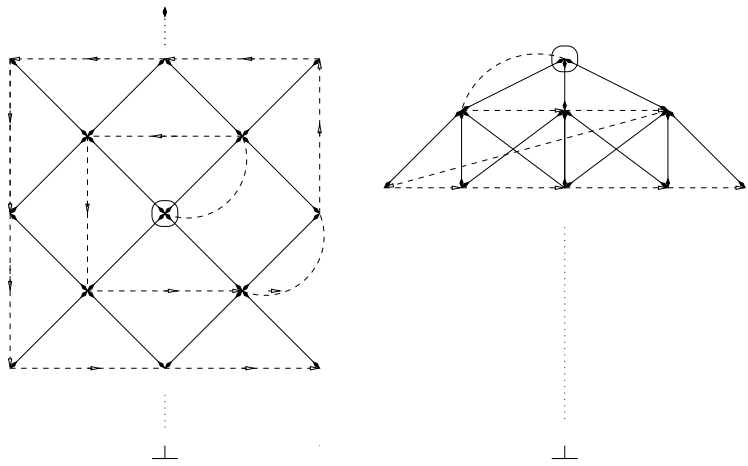


Fig. 2. A schematic visualization of the radial lattice search (left) compared to a top-down search (right). Nodes are explored starting at the encircled point and then following the dashed line. This view is simplified in that the employed heuristic function $h(C)$ is assumed to be constant for all C and descendants of explored nodes are inserted in the end of the *open* list, which in the top-down case corresponds to a breadth-first search.

bottom-up (visualized in Fig. 2). With this refinement operator, all nodes in the lattice can be reached from any starting node.

We shall now address the problem of choosing the seed, i.e. the initial stochastic clause selection. The principal difficulty of its implementation lies in devising a procedure for uniform random sampling of clauses from the search space. Here, we describe a procedure (from [14]) that does not require prior generation of all elements of the search space. Recall that these are definite clauses obtained from subsets of literals drawn from a most specific (definite) clause $\perp$. Additional provisos are that each subset is of cardinality at most $c+1$ (where c is a user-specified maximum number of negative literals) and is in the language $\mathcal{L}$. Let $\mathcal{C}$ denote all such clauses. Further, let the number of clauses in $\mathcal{C}$ with exactly l literals be n_l and $\mathcal{N}$ denote the subset of natural numbers $\{1,\ldots,|\mathcal{C}|\}$. Define a function $h : \mathcal{C} \to \mathcal{N}$ such that $h(C) = \sum_{i=1}^{|C|-1} n_i + j$ where $|C|$ is the number of literals in C and $1 \leq j \leq n_{|C|}$. That is, h provides a sequential enumeration of clauses by length. While many functions fit this requirement (depending on the enumeration adopted), it is easy to show that any such h is both $1-1$ and *onto*. It follows that h is invertible – that is, given a number in $\mathcal{N}$, it is possible to find a unique clause in $\mathcal{C}$ provided the n_i (and c) are known. In principle, it is therefore possible to achieve the selection required by randomly choosing a number n in $\mathcal{N}$ and returning $C = h^{-1}(n)$. Such an inverse function works as follows. Given a number $n > 0$: (a) find the largest number $l = 0 \ldots c$ such that $j = n - \sum_{i=0}^{l} n_i > 0$; (b) generate a sequence of clauses in $\mathcal{L}$ of length $l+1$. C is the j^{th} clause in this sequence. If n is randomly generated, then the clause

estimate$(\bot, \mathcal{L}, l, s)$: Given a most specific clause $\bot$ and a clause length $l > 1$, returns an estimate of the number of definite clauses of length l in $\mathcal{L}$ such that each clause is a subset of $\bot$. The estimate is obtained from a sample of size s.

1. Sample s clauses of length l from $\bot$. Each such clause consists of the positive ("head") literal in $\bot$ and a random selection, without replacement, of $l-1$ literals from the negative ("body") literals in $\bot$.
2. Determine the proportion p_l of the s clauses that are in $\mathcal{L}$.
3. return $p_l \times (|\bot| - 1) \times \ldots \times (|\bot| - l + 1)$

Fig. 3. A procedure for estimating the number of "legal" clauses of length $l > 1$. The estimate obtained in Step 3 above is unbiased [18]. The value of the sample size s needs to be decided. An option is to be guided by statistical estimation theory. This states that if values of p_l are not too close to 0 or 1, then we can be at least $100 \times (1-\alpha)\%$ confident that the error will be less than a specified amount e when $s = z^2_{\alpha/2}/(4e^2)$ [18]. Here z represents the standard normal variable as usual.

generation process does not have to be so, and can be made more efficient by various devices. Some examples are: (a) take C to be the first clause of that length (and in $\mathcal{L}$) that has not been drawn before; (b) a once-off generation of the appropriate number of clauses in $\mathcal{L}$ at each length ("appropriate" here means that the proportion of clauses of length i in the sample is $n_i/|\mathcal{N}|$); and (c) using a dependency graph over literals in $\bot$ to ensure that the random clause construction always results in clauses within the language $\mathcal{L}$.

In practice, without prior generation of the set $\mathcal{C}$, the n_i are not known for $i > 1$ and we adopt the procedure in Fig. 3 [3] for estimating them.

Having constructed a method for stochastic selection of the starting clause and its subsequent deterministic refinement, we are in a position to implement the technique of randomized rapid restarts. An implementation for the clause lattice search is described in Fig. 4. The underlying principle that makes the technique applicable to the clause search is that, unlike in usual ILP approaches, we do not search through a specified number of nodes, returning the best clause found, but rather we stop the search once a clause is found meeting a pre-specified condition of 'goodness' as follows.

$$pos(C) > 1 \tag{4}$$

$$\frac{pos(C)}{pos(C) + neg(C)} > Acc \tag{5}$$

where $pos(C)$ ($neg(C)$) is the number of positives (negative) examples covered by C. The first condition avoids the trivial solution $C = e$, $e \in E^+$ and the second condition is parameterized by a constant Acc.

We set the maximum number of allowed restarts $maxtries$, which should theoretically be infinite, to a finite number ($maxtries = 10$) because we cannot

[3] This method is implemented in the ILP system Aleph [6] and can serve as well for other methods of randomized local search, such as GSAT or WSAT.

$rrr(Lat, Acc, B, E^+, E^-, maxtime, maxtries)$: Given background knowledge B, positive and negative examples E^+, E^-, return a clause from the given subsumption lattice Lat, that satisfies the conditions in Eqs. 4 and 5 for the given constant Acc.

1. $tries := 1$
2. Select a random starting clause C_0 using the procedure described in Section 3.
3. $searchtime := 0$, start timing.
4. Starting at C_0, perform an exhaustive radial search described in Section 3, until $searchtime > maxtime$ or a clause C satisfying Eqs. 4, 5 is found.
5. If C was found, STOP, return C.
6. If $tries < maxtries$, $tries := tries + 1$, Go to 2. Otherwise return "failure".

Fig. 4. An implementation of randomized rapid restarts for the clause lattice search. The maximum time for one exhaustive search is limited by the constant $maxtime$. The maximum number of repeated searches is $maxtries$.

guarantee exclude that a clause satisfying the pre-specified condition exists in the lattice. In the case of reaching the limit $maxtries$ ("failure" in Fig. 4), the positive example used to construct the current bottom clause $\perp$ is returned as the resulting clause. The setting of $maxtime$ will be discussed in connection with the experimental setting in Section 4.

Finally, to cover all of the positive examples in the training sets of the experiments, we use the greedy covering approach as usual in ILP systems, i.e. the procedure in Fig. 4 is run repeatedly with a bottom clause constructed using one selected positive, until all positives are covered, each time adding the newly constructed clause to the background knowledge and deleting the covered positives from the training set.

4 Experiments

4.1 The Aim

We shall investigate (a) if the heavy-tailed phenomenon manifests itself when searching the subsumption lattice; and (b) if utilizing RRR will help improve search efficiency for problems exhibiting the heavy-tailed phenomenon.

4.2 Materials

Our experimental material consists of two sets of pre-classified data, namely the data on the mutagenic and carcinogenic properties of some chemicals. These data are publicly available (anonymous ftp to `ftp.comlab .ox.ac.uk` in the directories `pub/Packages/ILP/Datasets/mutagenesis/aleph` and `pub/Packages /ILP/Datasets/carcinogenesis/aleph`).
We refer the reader to [16,17] for detailed descriptions of background knowledge available for the mutagenesis task. The background information is encoded in approximately 13,000 facts. The background knowledge for the carcinogenesis

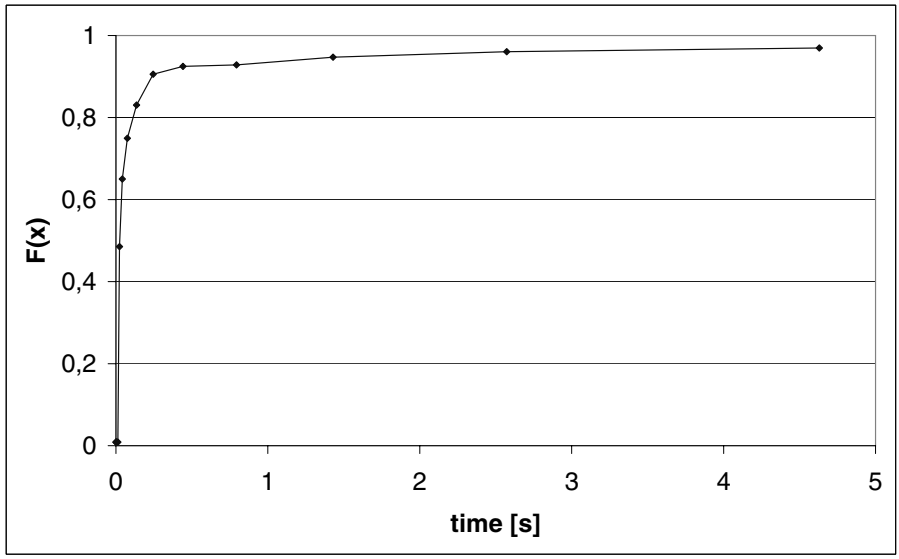


Fig. 5. The cumulative distribution $F(x)$ of runtimes of the randomized algorithm searching for a 'good' clause in the mutagenesis problem.

problem is conceptually of a similar nature. The encoding requires approximately 24,500 facts – see [15] for more details.

All of our subsequent experiments use an Athlon 1500MHz CPU – based computer with 512KB of RAM and the ILP program Aleph (Version 3). Aleph is available at: `http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.pl`.

The language $\mathcal{L}$ will be limited to clauses of maximum number of 4 negative literals and maximum variable depth [9] 2.

4.3 Methods and Results

We shall observe the stochastic behaviour of the randomized search-algorithm described in the previous section, namely the distribution of search-times required to find a clause C which satisfies the conditions in Eqs. 4,5, where we set $Acc = 0.7$.

Figure 5 shows the cumulative distribution $F(x)$ of runtimes for the mutagenesis task, Figure 6 an analogous distribution for the carcinogenesis task. Both of the distributions are collected from about 35,000 searches starting in random seeds. In the mutagenesis task, for example, almost 20% of runs arrive at a solution in less then 0.1s, however almost 30% of runs do not find a solution in 20s.

Figures 7 and 8 clearly show that both of the experimentally measured distributions exhibit a heavy-tail (c.f. Section 2). According to the study [4], our findings justify the application of the method of randomized rapid restarts. To

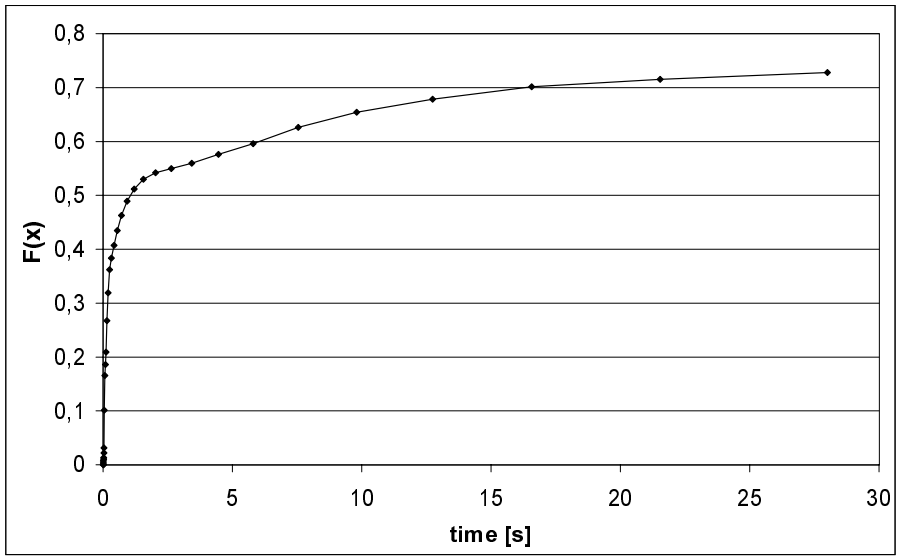


Fig. 6. The cumulative distribution $F(x)$ of runtimes of the randomized algorithm searching for a 'good' clause in the carcinogenesis problem.

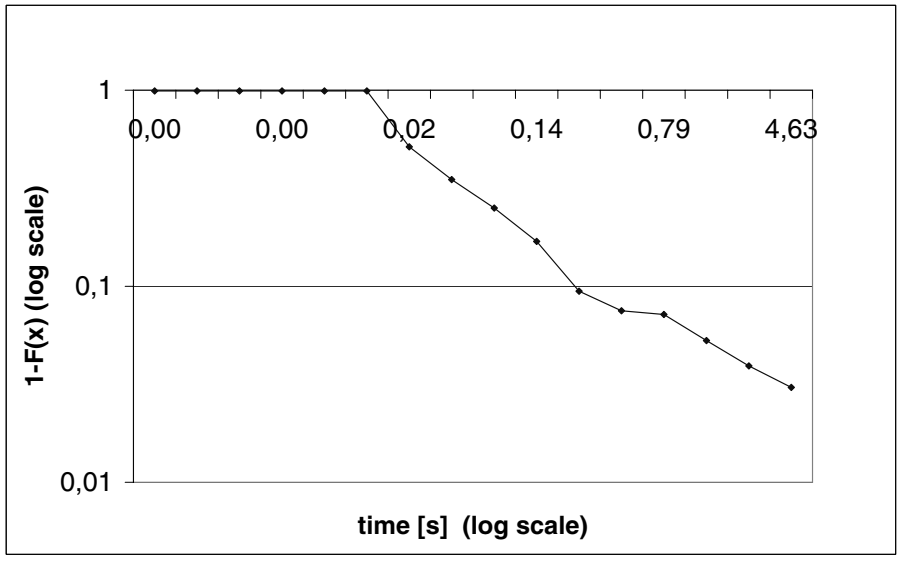


Fig. 7. A log-log plot of the distribution decay $1 - F(x)$ concerning the same data as in Figure 5.

use the algorithm described in Section 3, we need to set the cutoff value $maxtime$. There is no analytic way of determining the optimal value $maxtime_{opt}$ of the cutoff value, but it is reported [4] to lie below the median point of the runtime

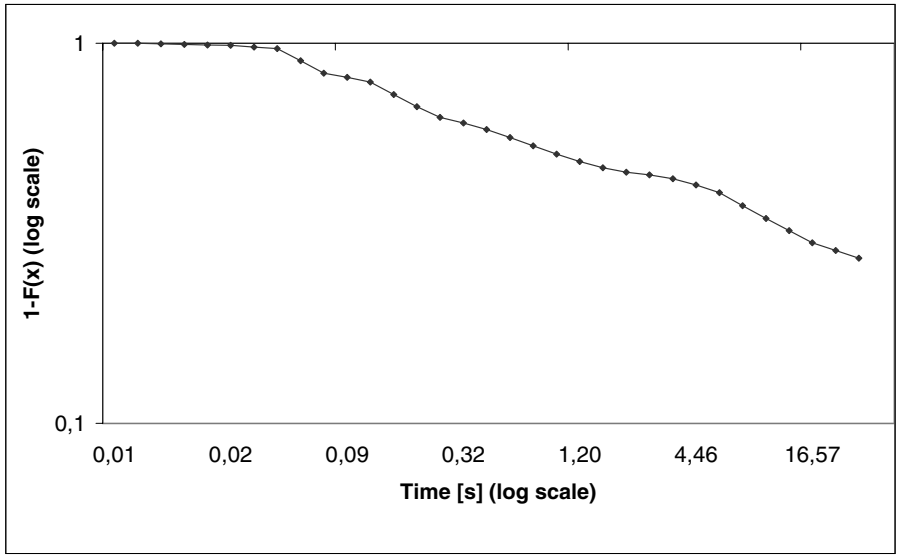


Fig. 8. A log-log plot of the distribution decay $1 - F(x)$ concerning the same data as in Figure 6.

distribution $F(x)$, i.e. $maxtime_{opt} << 1s$ for both our experimental domains. As it may be infeasible in the general case to construct the runtime distributions $F(x)$ for a given problem prior to the learning process, we shall disregard the information provided by the already generated distributions, and we choose a small *ad hoc* value $maxtime = 1s$ for both of the domains in the comparative experiments.

Table 1 summarizes the predictive accuracies and learning times of the randomized rapid restarts technique vs. the standard exhaustive breadth-first top-down search algorithm. The former method was tested with the *Acc* parameter set to the values 0.7 and 0.9. Similarly, the latter method was tested with two values 0.7 and 0.9 of the *minimum accuracy* requirement on a clause to be accepted for the constructed theory.

The results suggest that by using randomized rapid restarts we achieved a drastic reduction of the search times for the price of only a small loss in predictive accuracy.

5 Related Work

Besides the direct inspiration by the findings due to Gomes et al., this work is also related to the research of the *phase transition* effect. Phase transition has been observed in algorithms for solving difficult computational problems, namely NP-complete ones such as the constraint satisfaction problem (CSP) [13]. A *constraint tightness* parameter $p \in < 0; 1 >$ can be calculated for any CSP instance.

Table 1. Estimated predictive accuracies (A) and theory construction times (T). The entries are from a 10-fold cross-validation design with time entries rounded up to the nearest second. The numbers in parentheses are estimates of standard deviation. These are obtained by a simple binomial formula that ignores the dependencies across cross-validation runs. Exact calculation of standard deviations for results from cross-validation designs is confounded by these dependencies but the approximation used here has been found to be adequate (see [1], pg 307). The algorithms result from two search techniques employed by Aleph, namely: deterministic top-down (DTD) (with minimum clause accuracy setting 0.7 and 0.9, respectively) and randomized rapid restarts (with clause threshold 0.7 and 0.9, respectively). MUT refers to the mutagenesis problem, CANC to carcinogenesis. The search space is limited by the maximum clause-length of 5 literals and maximum variable depth 2.

Algorithm	MUT		CANC	
	A (%)	T (s)	A (%)	T (s)
DTD 0.7	88.76	1589	57.91	24092
	(5.99)	(461)	(9.75)	(11915)
DTD 0.9	88.23	1541	56.22	22101
	(5.63)	(459)	(8.98)	(9811)
RRR 0.7	87.71	9	54.84	74
	(7.62)	(4)	(8.97)	(10)
RRR 0.9	86.31	24	57.57	126
	(8.67)	(10)	(6.39)	(71)

According to empirical studies [13], the expected time to solve a CSP is small for values of p close to 0 (phase of 'many solutions available') or 1 (phase of 'inspecting a small search tree') and grows dramatically for p close to a *critical value* p_{cr} (transition between the two phases). In the surrounding of p_{cr}, the costs of solving CSP instances not only show a high mean, but also a large variability. Frost el al. [2] approximate the cost distributions of instances with p close to p_{cr} with various closed-form distributions. They point out (independently of Gomes el al.) the long tails of these distributions and report that "problems that are not solved early are likely to take a long time". The fundamental bridge between such findings and ILP is the fact that the first-order subsumption problem can be mapped onto a CSP [7]. Botta et al. then show [8] that a typical ILP program (FOIL) tends to "induce hypotheses generating matching problems located inside the phase transition region"; Giordana and Saitta report a similar observation [3] in real-world domains, including mutagenesis. Combining the referred results, the heavy-tailed effect had been expectable before our study, which can thus be seen as an empirical verification of this implicit expectation. Unlike the previous studies where statistics were measured for a collection of the proving (subsumption check) problem instances, we measured distributions on a collection of complete hypothesis-searching cycles, each containing a number of subsumption tests.

The way statistical observations are exploited towards efficiency improvements also distinguishes our approach from the mentioned related work. Se-

bag and Rouveirol [11] apply a stochastic algorithm in order to accelerate the subsumption-test and Giordana and Saitta [3] develop an on-line complexity estimator which can potentially be used for the same purpose. Our approach, on the other hand, allows to adopt the RRR technique to reduce the complexity of the hypothesis search in its entirety.

As far as the randomized technique of traversing through the subsumption lattice is concerned, to our best knowledge, there is only indirectly related work to our study. Serra et al [12] show that starting the search for a hypothesis from random seed formulas, instead than top-down, can be beneficial. Randomized search in an ILP system has been assessed in [14][4].

6 Conclusions

Our study has shown that the phenomenon of heavy-tailed runtime distributions occurs in two important experimental domains of ILP and we believe that it is typical to many other domains. Testing this hypothesis is a part of our future work.

This observation lead to the utilization of the technique of randomized rapid restarts which we reformulated for sakes of ILP. To apply this method, the exhaustive lattice search was randomized in such a way that we selected randomly the clause where the search was started. Randomized rapid restarts may then be used to reduce the average time required to find a clause with desired properties. A natural question is whether reducing the average runtime of the search procedure randomized in this way may lead to outperforming the deterministic top-down or bottom-up search. But clearly, if we do not impose a prior probability distribution on clauses (or e.g. on clause-lengths), there is no reason to expect that a search starting from the most general (most specific) element will be systematically faster than the average search starting in a random element of the lattice.

Using the technique of randomized rapid restarts, we were able to significantly reduce the search times in large hypothesis spaces of both of the tested domains.

Acknowledgements

We thank the ILP'02 referees for pointing us to some very relevant articles. Also, the ILP'02 audience contributed much to the paper by motivating us to relate our study to the phase transition research. Filip Zelezny greatly acknowledges the support from the EU project INCO 977102 ILPnet2 and the Czech Technical University grant CTU 0209013. David Page was supported by the U.S. National Science Foundation grant 9987841 and a U.S. DARPA EELD grant.

[4] We are also aware of the talk of Stephen Muggleton at the Machine Intelligence workshop in 2001 about randomization techniques in Progol but as we gather, there is no written account on that talk.

References

1. L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.
2. Daniel Frost, Irina Rish, and Lluis Vila. Summarizing CSP hardness with continuous probability distributions. In *AAAI/IAAI*, pages 327–333, 1997.
3. A. Giordana and L. Saitta. Phase transitions in relational learning. *Machine Learning*, 2000.
4. C. P. Gomes, B. Selman, N. Crato, and H. A. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1/2):67–100, 2000.
5. C. P. Gomes, B. Selman, and H. A. Kautz. Boosting combinatorial search through randomization. In *AAAI/IAAI*, pages 431–437, 1998.
6. http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html.
7. J. Maloberti and M. Sebag. Theta-subsumption in a constraint satisfaction perspective. volume 2157 of *Lecture Notes in Artificial Intelligence*, pages 164–178. Springer-Verlag, September 2001.
8. M.Botta, A.Giordana, L.Saitta, and M.Sebag. Relational learning: hard problems and phase transitions. In *6th Congress of the Italian Association for Artificial Intelligence*. Springer-Verlag, 1999.
9. S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
10. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.
11. Michele Sebag and Celine Rouveirol. Resource-bounded relational reasoning: Induction and deduction through stochastic matching. *Machine Learning*, 38(1/2):41–62, January 2000.
12. A. Serra, A. Giordana, and L. Saitta. Learning on the phase transition edge. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 921–926. Morgan Kaufmann, 2001.
13. Barbara M. Smith and Martin E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1-2):155–181, 1996.
14. A. Srinivasan. A study of two probabilistic methods for searching large spaces with ILP. Technical Report PRG-TR-16-00, Oxford University Computing Laboratory, Oxford, 2000.
15. A. Srinivasan and R. King. Carcinogenesis predictions using ilp. In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Artificial Intelligence*, pages 3–16. Springer-Verlag, 1997.
16. A. Srinivasan and R.D. King. Feature construction with Inductive Logic Programming: a study of quantitative predictions of biological activity aided by structural attributes. *Data Mining and Knowledge Discovery*, 3(1):37–57, 1999.
17. A. Srinivasan, S. Muggleton, M.J.E. Sternberg, and R.D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1,2), 1996.
18. R. Walpole and R. Myers. *Probability and Statistics for Engineers and Scientists*. Collier Macmillan, New York, 1978.

Learning in Rich Representations: Inductive Logic Programming and Computational Scientific Discovery*

Sašo Džeroski

Department of Intelligent Systems, Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
Saso.Dzeroski@ijs.si

The goals of this presentation are as follows:

- Review some key ideas and developments in inductive logic programming.
- Show how these ideas can be used in other learning settings, and in particular for the computational scientific discovery of quantitative laws.
- Encourage more research on learning in rich representations, such as relational representations and differential equations, which can be used for modeling a variety of real world problems.

The slides for this presentation can be found at
http://www-ai.ijs.si/SasoDzeroski/ICML02/

Inductive logic programming (ILP) is concerned with learning from data and domain knowledge in relational representations. ILP started off by addressing the task of learning logic programs from examples and background knowledge (Muggleton 1992; Lavrač and Džeroski 1994; De Raedt 1996). Recent developments, however, have broadened its scope to address a variety of learning tasks in relational representations. A significant part of ILP research now goes under the heading (Multi)Relational Data Mining – (M)RDM (Džeroski and Lavrač 2001) – and is concerned with finding patterns such as relational association rules and relational decision trees from multi-table relational databases. As another example, ILP has been also used in a reinforcement learning context (Džeroski et al. 1998; 2001; Driessens and Džeroski 2002).

Key ideas from ILP include:

- Transforming ILP problems to propositional form (Lavrač and Džeroski 1994);
- The use of background knowledge;
- Refinement operators (Shapiro 1983);
- Declarative language bias (Nedellec et al. 1996);
- Theory revision (De Raedt 1992; Wrobel 1996).

These ideas have received significant attention within ILP, but are not specific to it. We have successfully used most of these ideas also within the context of computational scientific discovery of quantitative laws.

* This paper also appears in the Proceedings of the Nineteenth Conference on Machine Learning (ICML- 2002), published by Morgan Kaufmann.

S. Matwin and C. Sammut (Eds.): ILP 2002, LNAI 2583, pp. 346–349, 2003.

Computational scientific discovery (Langley et al. 1987; Shrager and Langley 1990) is concerned with applying computational methods to automate scientific activities. Early research on computational discovery (Langley et al. 1987) focussed on reconstructing episodes from the history of science by modeling the scientific activities and processes that led to the scientist's insight. Recent efforts in this area (for overviews see Langley 2000; Džeroski and Todorovski 2002) have focussed on individual scientific activities (such as formulating quantitative laws). Much of the work in computational scientific discovery has put emphasis on formalisms used to communicate among scientists, including numeric equations, structural models, and reaction pathways.

Over the last decade, we have developed a number of approaches to discovering quantitative laws in the form of algebraic, ordinary differential (ODEs) and partial differential (PDEs) equations (Džeroski and Todorovski 1993; Todorovski and Džeroski 1997; Todorovski et al. 2000). They have been applied to a number of practical modeling problems, mainly in the area of ecology (Todorovski et al. 1998; Džeroski et al. 1999). We have devoted special attention to the use of various forms of domain knowledge: we use declarative bias (Todorovski and Džeroski 1997) and background knowledge (Džeroski and Todorovski 2001; Langley et al. 2002), and also address the problem of revising theories that consist of quantitative laws (Todorovski and Džeroski 2001). A number of key ideas from ILP were used in these developments:

- Introducing ordinary and partial derivatives by numerical derivation and transforming the problem of discovering ODEs and PDEs to the problem of discovering algebraic equations.
- Using declarative bias and refinement operators to define and search the space of equations.
- Revising quantitative laws in the light of new observations, trying to retain as much as possible of the originals laws.

Learning in rich representations (such as relational representations and differential equations) allows for a realistic approach to learning in difficult domains. Rather than trying to solve a difficult problem by starting from scratch, one can use existing domain knowledge in addition to collected observations (examples) and build upon it. Different types of domain knowledge can be taken into account, such as concepts already in common use (background knowledge), intuitions about the form of the target theory (declarative bias) and existing theories (theory revision). In this context, one can trade-off between the quantity and quality of observations and domain knowledge: high quantities of quality data may suffice to generate a good theory even with no domain knowledge, while smaller quantities of (lower quality) data may suffice if relevant domain knowledge is available. We believe this is of great importance and would like to encourage further research on learning in rich representations.

References

1. De Raedt, L., editor (1996). *Advances in Inductive Logic Programming*. IOS Press, Amsterdam.
2. De Raedt, L. (1992). *Interactive Theory Revision: An Inductive Logic Programming Approach*. Academic Press, London.
3. Driessens, K., & Džeroski, S. (2002). Integrating experimentation and guidance in relational reinforcement learning. In *Proc. 19th International Conference on Machine Learning* (pp. 115–122). Morgan Kaufmann, San Francisco, CA.
4. Džeroski, S., & Todorovski, L., editors (2002). *Computational Discovery of Communicable Knowledge*. Springer, Berlin. Forthcoming.
5. Džeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, 43: 7–52.
6. Džeroski, S., & Lavrač, N., editors (2001). *Relational Data Mining*. Springer, Berlin.
7. Džeroski, S., & Todorovski, L. (2001). Encoding and using domain knowledge on population dynamics in equation discovery. In L. Magnani, N. J. Nersessian, and C. Pizzi, (editors), *Logical and Computational Aspects of Model-Based Reasoning*. Kluwer, Dordrecht. In press.
8. Džeroski, S., Todorovski, L., Bratko, I., Kompare, B., & Križman, V. (1999). Equation discovery with ecological applications. In A.H. Fielding, editor, *Machine Learning Methods for Ecological Applications* (pp. 185–207). Kluwer, Dordrecht.
9. Džeroski, S., De Raedt, L., & Blockeel, H. (1998). Relational reinforcement learning. In *Proc. 15th International Conference on Machine Learning* (pp. 136–143). Morgan Kaufmann, San Francisco, CA.
10. Džeroski, S., & Todorovski, L. (1993). Discovering dynamics. In *Proc. 10th International Conference on Machine Learning* (pp. 97–103). Morgan Kaufmann, San Mateo, CA.
11. Langley, P., Sanchez, J., Todorovski, L., & Džeroski, S. (2002). Inducing process models from continuous data. In *Proc. 19th International Conference on Machine Learning* (pp. 347–354). Morgan Kaufmann, San Francisco, CA.
12. Langley, P. (2000). The computational support of scientific discovery. *International Journal of Human-Computer Studies*, 53: 393–410.
13. Langley, P., Simon, H.A., Bradshaw, G.L., & Zytkow, J. (1987). *Scientific Discovery*. MIT Press, Cambridge, MA.
14. Lavrač, N., & Džeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, Chichester. Freely available at `http://www-ai.ijs.si/SasoDzeroski/ILPBook/`.
15. Muggleton, S., editor (1992). *Inductive Logic Programming*. Academic Press, London.
16. Nedellec, C., Rouveirol, C., Ade, H., Bergadano, F., & Tausend, B. (1996). Declarative bias in inductive logic programming. In (De Raedt 1996) (pp. 82–103).
17. Shapiro, E. (1983). *Algorithmic Program Debugging*. MIT Press, Cambridge, MA.
18. Shrager, J., & Langley, P., editors (1990). *Computational Models of Scientific Discovery and Theory Formation*. Morgan Kaufmann, San Mateo, CA.
19. Todorovski, L., & Džeroski, S. (2001). Theory revision in equation discovery. In *Proc. 4th International Conference on Discovery Science* (pp. 390–400). Springer, Berlin.

20. Todorovski, L., Džeroski, S., Srinivasan, A., Whiteley, J., & Gavaghan, D. (2000). Discovering the structure of partial differential equations from example behavior. In *Proc. 17th International Conference on Machine Learning* (pp. 991–998). Morgan Kaufmann, San Francisco, CA.
21. Todorovski, L., Džeroski, S., & Kompare, B. (1998). Modeling and prediction of phytoplankton growth with equation discovery. *Ecological Modelling* 113: 71–81.
22. Todorovski, L., & Džeroski, S. (1997). Declarative bias in equation discovery. In *Proc. 14th International Conference on Machine Learning* (pp. 376–384). Morgan Kaufmann, San Francisco, CA.
23. Wrobel, S. (1996). First order theory refinement. In (De Raedt 1996) (pp. 14–33).

Author Index