Fabrizio Riguzzi Filip Zelezny

ILP 2012
Late Breaking Papers

Late Breaking Papers from the 22nd International Conference on Inductive Logic Programming (ILP 2012)

Dubrovnik, Croatia, September 17-19, 2012

CEUR-WS.org/Vol-975

http://ceur-ws.org/Vol-975/

urn:nbn:de:0074-975-8

Editors' addresses:
Fabrizio Riguzzi
Department of Mathematics and Computer Science, University of Ferrara
Via Saragat 1, 44122, Ferrara, Italy

fabrizio.riguzzi@unife.it

Filip Zelezny
Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague
Karlovo namesti 13, 12135 Prague 2, Czech Republic

zelezny@fel.cvut.cz

Preface

This volume contains the Late Breaking Papers of ILP 2012: the 22nd International Conference on Inductive Logic Programming held on September 17-19, 2012 in Dubrovnik.

The ILP conference series, started in 1991, is the premier international forum on learning from structured data. Originally focusing on the induction of logic programs, it broadened its scope and attracted a lot of attention and interest in recent years. The conference now focuses on all aspects of learning in logic, multi-relational learning and data mining, statistical relational learning, graph and tree mining, relational reinforcement learning, and other forms of learning from structured data.

This edition of the conference solicited three types of submissions:

1. long papers (12 pages) describing original mature work containing appropriate experimental evaluation and/or representing a self-contained theoretical contribution.
2. short papers (6 pages) describing original work in progress, brief accounts of original ideas without conclusive experimental evaluation, and other relevant work of potentially high scientific interest but not yet qualifying for the above category.
3. papers relevant to the conference topics and recently published or accepted for publication by a first-class conference such as ECML/PKDD, ICML, KDD, ICDM etc. or journal such as MLJ, DMKD, JMLR etc.

We received 20 long and 21 short submissions, and 1 previously published paper. Each submission was reviewed by at least 3 program commitee members. The short papers were evaluated on the basis of both the submitted manuscript and the presentation at the conference. Accepted papers presenting work in progress, i.e., reports on ongoing research are collected in this volume.

The conference program included 3 invited talks. In the lecture entitled *Declarative Modeling for Machine Learning*, Luc De Raedt proposed to apply the constraint programming methodology to machine learning and data mining and to specify machine learning and data mining problems as constraint satisfaction and optimization problems. In this way it is possible to develop applications and software that incorporates machine learning or data mining techniques by specifying declaratively what the machine learning or data mining problem is rather than having to outline how the solution needs to be computed.

Ben Taskar's talk *Geometry of Diversity and Determinantal Point Processes: Representation, Inference and Learning* discussed approaches to inference and learning in graphical models using determinantal point processes (DPPs) that offer tractable algorithms for exact inference, including computing marginals, computing certain conditional probabilities, and sampling. He presented recent work on a novel factorization and dual representation of DPPs that enables efficient inference for exponentially-sized structured sets.

Geraint A. Wiggins spoke about *Learning and Creativity in the Global Workspace* and presented a model based on Baars Global Workspace account of consciousness, that attempts to provide a general, uniform mechanism for information regulation. The key ideas

involved are: information content and entropy, expectation, learning multi-dimensional, multi-level representations and data, and data-driven segmentation. The model was originally based in music, but can be generalised to language. Most importantly, it can account for not only perception and action, but also for creativity, possibly serving as a model for original linguistic thought.

The conference was kindly sponsored by the Office of Naval Research Global, the Artificial Intelligence journal and the Machine Learning journal. We would like to thank Easychair.org for supporting submission handling. Our deep thanks go also to Nada Lavrač, Tina Anžič and Dragan Gamberger for the local organization of the conference and Radomír Černoch for setting up and maintaining the conference web site.

March 21, 2013
Ferrara, Prague

Fabrizio Riguzzi
Filip Zelezny

Program Committee

Erick Alphonse	LIPN - UMR CNRS 7030
Dalal Alrajeh	Imperial College London
Annalisa Appice	University Aldo Moro of Bari
Ivan Bratko	University of Ljubljana
Rui Camacho	LIACC/FEUP University of Porto
James Cussens	University of York
Saso Dzeroski	Jozef Stefan Institute
Floriana Esposito	Dipartimento di Informatica, Università di Bari
Nicola Fanizzi	Dipartimento di Informatica, Università di Bari
Daan Fierens	Katholieke Universiteit Leuven
Nuno Fonseca	CRACS-INESC Porto LA
Tamas Horvath	University of Bonn and Fraunhofer IAIS
Katsumi Inoue	National Institute of Informatics
Nobuhiro Inuzuka	Nagoya Institute of Technology
Andreas Karwath	University of Freiburg
Kristian Kersting	Fraunhofer IAIS and University of Bonn
Ross King	University of Wales, Aberystwyth
Ekaterina Komendantskaya	School of Computing, University of Dundee
Stefan Kramer	TU München
Nada Lavrač	Jozef Stefan Institute
Francesca Alessandra Lisi	Università degli Studi di Bari "Aldo Moro"
Donato Malerba	Dipartimento di Informatica, Universita' di Bari
Stephen Muggleton	Department of Computing, Imperial College London
Ramon Otero	University of Corunna
Aline Paes	COPPE/PESC/UFRJ
David Page	University of Wisconsin, Madison, WI
Bernhard Pfahringer	University of Waikato
Ganesh Ramakrishnan	IIT Bombay
Jan Ramon	K.U.Leuven
Oliver Ray	University of Bristol
Fabrizio Riguzzi	University of Ferrara
Celine Rouveirol	LIPN, Universit Paris 13
Chiaki Sakama	Wakayama University
Jose Santos	Imperial College London
Vítor Santos Costa	Universidade do Porto
Michèle Sebag	Univ. Paris-Sud, CNRS
Jude Shavlik	University of Wisconsin – Madison
Takayoshi Shoudai	Department of Informatics, Kyushu University
Ashwin Srinivasan	Souht Asian University
Prasad Tadepalli	Oregon State University
Alireza Tamaddoni-Nezhad	Imperial College, London
Tomoyuki Uchida	Hiroshima City University

Christel Vrain	LIFO - university of Orléans
Stefan Wrobel	Fraunhofer IAIS - Univ. of Bonn
Akihiro Yamamoto	Kyoto University
Gerson Zaverucha	PESC/COPPE - UFRJ
Filip Zelezny	Czech Technical University in Prague

Additional Reviewers

D

Di Mauro, Nicola
Duboc, Ana Luisa
F

Ferilli, Stefano
S

Stein, Sebastian
T

Trajanov, Aneta

Contents

A Link-Based Method for Propositionalization
Quang-Thang Dinh, Christel Vrain and Matthieu Exbrayat **10**

MicroRNAs robustness in genetic regulatory networks
Andrei Doncescu, Katsumi Inoue and Jacques Demongeot **26**

A Problog Model For Analyzing Gene Regulatory Networks
Antonio Goncalves, Irene Ong, Jeffrey Lewis and Vitor Costa **38**

Creative Problem Solving by Concept Generation Using Relation Structure
Katsutosh Kanamori and Hayato Ohwada **44**

On Computing Minimal Generators in Multi-Relational Data Mining with respect to theta-Subsumption
Noriaki Nishio, Atsuko Mutoh and Nobuhiro Inuzuka **50**

A Wordification Approach to Relational Data Mining: Early Results
Matic Perovšek, Anže Vavpetič and Nada Lavrač **56**

Hybrid Logical Bayesian Networks
Irma Ravkic, Jan Ramon and Jesse Davis **62**

Towards an Automated Pattern Selection Procedure in Software Models
Alexander van Den Berghe, Jan Van Haaren, Stefan Van Baelen, Yolande Berbers and Wouter Joosen **68**

Non-monotone dualization via monotone dualization
Yoshitaka Yamamoto, Koji Iwanuma and Katsumi Inoue **74**

A Link-Based Method for Propositionalization

Quang-Thang DINH, Matthieu EXBRAYAT, Christel VRAIN

LIFO, Bat. 3IA, Université d'Orléans
Rue Léonard de Vinci, B.P. 6759, F-45067 ORLEANS Cedex 2, France
{Thang.Dinh,Matthieu.Exbrayat,Christel.Vrain}@univ-orleans.fr
http://www.univ-orleans.fr/lifo/

Abstract. Propositionalization, a popular technique in Inductive Logic Programming, aims at converting a relational problem into an attribute-value one. An important facet of propositionalization consists in building a set of relevant features. To this end we propose a new method, based on a synthetic representation of the database, modeling the links between connected ground atoms. Comparing it to two state-of-the-art logic-based propositionalization techniques on three benchmarks, we show that our method leads to good results in supervised classification.

1 Introduction

Propositionalization is a popular technique in ILP, that aims at converting a relational problem into an attribute-value one [1–5]. Propositionalization usually is decomposed into two main steps: generating a set of useful attributes (features) starting from relational representations and then building an attribute-value table, which can be mono-instance (a single tuple for each example) or multi-instance (several tuples for an example). Traditional attribute-value algorithms can then be applied to solve the problem. Approaches for constructing automatically the new set of attributes (features) can be divided into two trends [6, 7]: methods based on logic *or* inspired from databases.

The first trend follows the ILP tradition which is logic-based. This trend, as far as we know, includes the first representative LINUS system [8] and its descendants, the latest being RSD [9], HiFi [4] and RELF [5]. For these systems, examples are mostly represented as first-order Herbrand interpretations and features are conjunctions of first-order function-free atoms. The search for features is based on a *template* (a set of ground atoms of which all arguments fall in exactly one of two categories: "input" or "output") or mode declarations (defining the predicates and assigning a *type* and *mode* to each of their arguments).

The second trend is inspired from databases and appeared later beginning with systems like Polka [2], RELAGGS [10] and RollUp [7]. Those systems build attributes, which summarize information stored in non-target tables by applying usual database aggregate functions such as count, min, max, etc.

In this paper, we propose a new method, called *Link-Based Propositionalization* or LBP, to build features for propositionalization from a set of ground atoms, without information on templates or mode declarations. The method was

initially designed to learn the structure of Markov logic networks [11], where it was used as a strategy to build a boolean table and to find dependent literals. The originality of the method is to build an abstract representation of sets of connected ground atoms, allowing thus to represent properties between objects.

LBP differs from the classical logic-based approaches both in the *semantic of the boolean table* and in the *search for features*. For example, the RELF system uses a block-wise technique to construct a set of tree-like conjunctive relational features while the others, like HIFI or RSD use the traditional level-wise approaches. The search in LBP does not rely on template or mode declarations, but on a synthetic representation of the dataset, namely the links of the chains, which allows to build features as well as to construct the boolean table based on the regularities of these chains. The notion of chain is related to relational path-finding [12] and relational cliché [13].

Our propositional method is presented in Section 2. We present related works in Section 3. Section 4 is devoted to experiments and finally, Section 5 concludes this paper.

2 Link-Based Propositionalization

Given as input a database DB and a query predicate Q, we present here a heuristic Link Based Propositionalization method (LBP) in order to transform relational information in data into an approximative representation in form of a boolean table. Once this boolean table has been learnt, it can be used for several tasks: looking for the most frequent patterns satisfied by instances of predicate Q, looking for the most discriminative patterns satisfied by positive examples of Q, or as input of a propositional learner for learning a model classifying positive from negative examples.

2.1 Preliminary notions

Let us recall here some basic notions of first order logic. We consider a function-free first order language composed of a set $\mathcal{P}$ of predicate symbols, a set C of constants and a set of variables. An *atom* is an expression $p(t_1, \ldots, t_k)$, where p is a predicate and t_i are either variables or constants. A *literal* is either a positive or a negative atom; it is called a *ground literal* when it contains no variable and a *variable literal* when it contains only variables. A *clause* is a disjunction of literals. Two ground atoms are *connected* if they share at least a constant (or argument).

A *variabilization* of a ground clause e, denoted by $var(e)$, is obtained by assigning a new variable to each constant and replacing all its occurrences accordingly.

The method that we propose is based on an abstract representation of sets of connected atoms, either ground atoms or variable atoms. This abstract representation is learned from sets of connected ground atoms and it is used to build sets of connected variable literals. Let us first introduce this representation.

2.2 An abstract representation

The idea underlying this method is to detect regularities in ground atoms: we expect that many chains of connected atoms are similar, and could thus be variabilized by a single chain. The similarity between chains of ground atoms is captured by the notion of links that we introduce in this paper and that models the relations between connected atoms.

Definition 1. *Let* g *and* s *be two ground literals (resp. two variable literals). A* link *between* g *and* s *is a list composed of the name of the predicates of* g *and* s *followed by the positions of the shared constants (resp. variables). It is written* link(g, s) = {G S g^0 s^0 / g^1 s^1 / ... } *where* G *and* S *are the predicate symbols of* g *and* s, $g^i \in [1, arity(g)]$, $s^i \in [1, arity(s)]$ *and the combinations / g^i s^i / mean that the constants respectively at position g^i in* g *and s^i in* s *are the same. If* g *and* s *do not share any constant then* link(g,s) *is empty.*

We are interested in representing the properties of sets of connected literals. In order to have a sequential representation of these properties, we consider only chains of literals defined as follows:

Definition 2. *A* chain *of ground literals (resp. variable literals) starting from a ground (resp. variable) literal g_1 is a list of ground (resp. variable) literals $\langle g_1, ..., g_k, ...\rangle$ such that $\forall i > 1$, $link(g_{i-1}, g_i)$ is not empty and every constant (resp. variable) shared by g_{i-1} and g_i is not shared by g_{j-1} and $g_j, 1 < j < i$. It is denoted by $chain(g_1) = \langle g_1, ..., g_k, ...\rangle$. The length of the chain is the number of atoms in it.*
The link of the chain $gc = \langle g_1, ..., g_k, ...\rangle$ *is the ordered list of links $link(g_i, g_{i+1})$, $i \geq 1$, denoted by $link(gc) = \langle link(g_1, g_2)/.../link(g_i, g_{i+1})/...\rangle$. The link of a chain composed of a single atom is the empty list. When a chain is composed of only two atoms, its link is the link between its two atoms. A chain of ground literals (resp. variable literals) is called, for short, a* ground chain *(resp. a* variable chain*).*

Let us notice that in this definition, it is only require that the variable shared by g_{i-1} and g_i is not used in previous links. But there may exist in g_{i-1} or in g_i some constants occurring in g_j, $j < i - 1$. Sometimes, it may be useful to know if a link has been obtained from a chain of ground atoms or from a chain of variable literals. In such situations, the term *link* is prefixed by *g-*, for expressing that the link has been obtained by a ground chain or by *v-*.

Definition 3. *A link $\langle g_1, ..., g_k\rangle$ is said to be a prefix of another link $\langle s_1, ..., s_n\rangle$, if $link(g_i, g_{i+1}) = link(s_i, s_{i+1})$, $\forall i, 1 \leq i < k$.*

Example 1. Let *DB1 = {P(a, b), Q(b, a), R(b, c), S(b), S(c)}* be a set of ground atoms. *P(a, b)* and *Q(b, a)* are connected by the two shared constants *a* and *b*. The constant *a* occurs respectively at position *1* of the ground atom *P(a, b)* and at position *2* of the ground atom *Q(b, a)*. Similarly, the constant *b* occurs

respectively at position *2* of the ground atom *P(a, b)* and at position *1* of the ground atom *Q(b, a)*. We have: *link(P(a, b), Q(b, a))* = $\{P\ Q\ 1\ 2\ /\ 2\ 1\}$.

A possible *chain* starting from the ground atom *P(a, b)* is $\langle$*P(a, b), R(b, c), S(c)*$\rangle$. Its link is $\langle\{P\ R\ 2\ 1\ \}\ /\ \{R\ S\ 2\ 1\}\rangle$. The link of the chain $\langle$*P(a, b), R(b, c)*$\rangle$ is $\langle\{P\ R\ 2\ 1\ \}\rangle$; it is a prefix of the previous link.

On the other hand, $\langle$*P(a, b), R(b, c), S(b)*$\rangle$ is not a chain as the constant b shared by *R(b, c)* and *S(b)* is already used to link *P(a, b)* and *R(b, c)*.

Definition 4. *A* variabilization of a link l *is a chain* c *of variable literals, so that* $link(c) = l$.

Let us for instance consider the link $\langle\{P\ Q\ 1\ 1\}\rangle$ where Q is a unary predicate. Then there is a single way, up to a renaming of variables, of variabilizing it, preserving the link, that is $P(A, B), Q(A)$. Nevertheless, given a link there may exist several ways of variabilizing it into a variable chain.

Let us now consider the link $\langle\{P\ R\ 1\ 2\}\ /\ \{R\ R\ 1\ 1\}\ \rangle$. This gives the scheme $P(slot_1, slot_2), R(slot_3, slot_4), R(slot_5, slot_6)$ with the constraints $slot_1 = slot_4$, $slot_1 \neq slot_3$, $slot_2 \neq slot_3$, $slot_2 \neq slot_4$ (thus satisfying the link $\{P\ R\ 1\ 2\}$) $slot_3 = slot_5$, $slot_3 \neq slot_6$, $slot_4 \neq slot_5$, $slot_4 \neq slot_6$ (for respecting the link $\{R\ R\ 1\ 1\}$) and the constraints $slot_3 \neq slot_4$ (for having a chain). Up to a renaming of variables, it can be variabilized into

$$P(X, Y), R(Z, X), R(Z, W) \text{ or into } P(X, Y), R(Z, X), R(Z, Y).$$

These two variabilizations correspond to two different strategies for filling the slots from left to right by variables, starting from the first atom composed of the first predicate and different variables as arguments. The first one consists in introducing a new variable each time there is no constraints on a slot of a predicate. The second one consists in trying to fill it with a variable that already exists, respecting the inequality constraints, thus leading to a more specific conjunction than the first one. This second strategy can still lead to several variabilizations, when several constants already introduced fulfill the constraints. The number of variabilizations can be reduced by using information on types of arguments when predicates are typed.

We define two strategies for variabilizing a link, and a third strategy for variabilizing a link, given the ground chain it comes from.

Definition 5. *Let P be the first predicate occurring in the link and let n be its arity. In both strategies, the first atom is $P(X_1, \ldots, X_n)$. Then slots are filled from left to right. For each slot with no equality constraints to fulfill:*

- *general variabilization: introduce a new variable*
- *specific variabilization: if possible, use a variable already introduced that fulfill all the inequalities constraints on this slot and the type of the argument.*
- *simple strategy: given a ground chain and its link, variabilize the ground chain, simply turning constants into variables.*

2.3 Creation of a set of features

Let us consider a target predicate Q and a training dataset *DB*. We aim at building a set of variable chains $\mathcal{F}$ linked to Q given *DB* such that for each

true ground atom A built with predicate Q in DB, and for each chain *chain(A)* starting from A, there exists a chain c in $\mathcal{F}$ such that $link(chain(A)) = link(c)$. It is reasonable to expect that many chains (starting from several ground atoms) are similar in the sense that their links are identical and could thus be variabilized by a single chain, with the same link.

The algorithm can be sketched as follows (in practice the length of the chains is limited by an integer k)

- for each ground atom A of the target predicate P,
  - find every chain starting from A
  - build the corresponding link and check whether it is a prefix of a link already built
  - if not, variabilize it.

In the current implementation, we have chosen the simple variabilization strategy, thus variabilizing the ground chain that has lead to the link under process. Moreover, we design two versions: in the first one (called LBP+), only positive ground atoms are considered, in the second one (called LBP-), positive and negative ground examples are considered.

Example 2. Let DB be a database composed of 14 ground atoms as follows: *advisedBy(bart, ada), student(bart), professor(ada), publication(t1,bart), publication(t2, bart), publication(t1, ada), publication(t2, ada), advisedBy(betty, alan), student(betty), professor(alan), publication(t3, betty), publication(t3, alan), publication(t3, andrew), professor(andrew).*

Figure 1 illustrates the production of chains of ground literals with the corresponding links and the resulting variable chains, using *advisedBy* as the target predicate, and bounding the length of chains to 4.

Starting from $advisedBy(bart, ada)$, several chains of ground literals can be built. For each chain, its link is built. For instance, the first chain built is {*advisedBy(bart, ada) student(bart)*}, leading to the link {*advisedBy student 1 1*}. This link is stored in the Set of Links and a first variable chain is created from this link.

The second chain {*advisedBy(bart, ada)*, *publication(t1, bart)*, *publication(t1, ada)*, *publication(t2,ada)*} leads to the link ⟨{*advisedBy publication 1 2*} / {*publication publication 1 1*}/ {*publication publication 2 2*}⟩. This link is not a prefix of the previous link, it is stored and a new variable chain is built from this link. Let us insist here on the fact that this chain depends on the variabilization strategy. The *general strategy* would lead to the chain {*advisedBy(A, B)*, *publication(C, A)*, *publication(C, D)*, *publication(E,D)*}. The *specific strategy* first introduces the new variable C as first argument of the first occurrence of *publication* (it cannot be equal to A or to B, otherwise it would have been written in the first link), leading to {*advisedBy(A, B)*, *publication(C, A)*}. When considering the second occurrence of publication, its first argument is given by the link. For its second argument, since no equality constraint is given on it, instead of introducing a new variable, it tries to use a previous variable: it cannot be A (it would have been given in the link), therefore it chooses B, leading to {*advisedBy(A, B)*, *publication(C, A)*, *publication(C,B)*}. Finally the third occurrence of *publication*

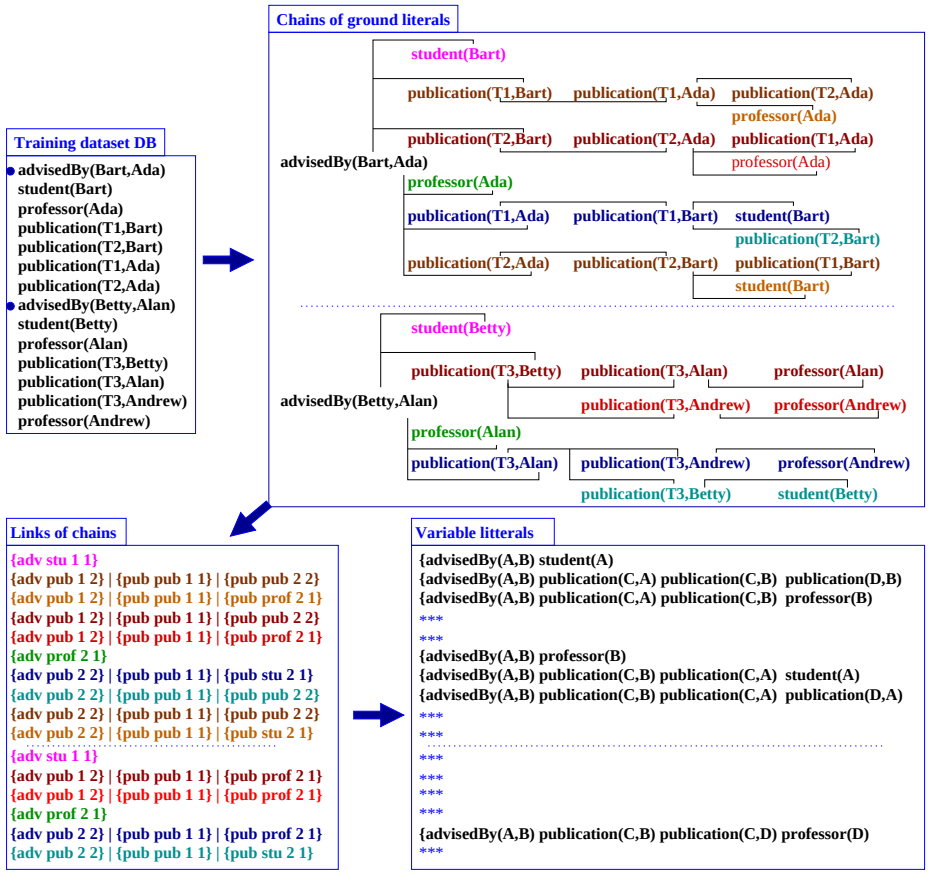


Fig. 1. The variabilization process using *chains* and *links* (length ≤ 4)

is introduced: its second argument is given by the link, no constraint is given on the first argument, no previous constant can be used (A and B do not have the same type and C cannot be used, because of the link. Thus we get {*advisedBy(A, B)*, *publication(C, A)*, *publication(C,B)*, *publication(D,B)*}. Since we know the ground chain that has lead to this link, the third strategy, called *simple strategy* variabilizes the ground chain, simply turning constants into variables. In this specific case, it leads to the same variable chain as the specific strategy. The simple strategy is the one we choose in the current implementation.

The third chain {*advisedBy(bart, ada)*, *publication(t1, bart)*, *publication(t1, ada)*, *professor(ada)*} leads to the link $\langle$ {*advisedBy publication 1 2*} / {*publication publication 1 1*}/ {*publication professor 2 1*}$\rangle$. This link is not a prefix of the previous link, it is stored and a new variable chain is built from this link leading to {*advisedBy(A, B)*, *publication(C, A)*, *publication(C, B)*, *professor(B)*}.

The process goes on. For instance, the chain {*advisedBy(bart, ada)*, *publication(t2, bart)*, *publication (t2, ada)*, *publication(t1,ada)*} leads to the same link as the second chain, and it is not kept.

The three stars sign (***) displayed in Figure 1 means that there is no new variabilization for the corresponding chain. As can be seen, at the end, there are 16 ground chains starting from ground atoms built with *advisedBy* but only 7 different links and therefore 7 variable chains. Let us notice that the notion of chains allows to capture interesting relations, as for instance the relation between $advisedBy(A, B)$ and the fact that A and B have a common publication.

Let us notice that for improving the efficiency of the implementation, types information on predicates are used, as for instance *professor(person)*, *student (person)*, *advisedBy(person, person)*, *publication(title, person)*.

2.4 Creating a Set of Variable Literals and a Boolean Table

In classical approaches for propositionalization, the chains that have been built become features. Here, we intend to benefit from the structure of the chains. For instance, if we consider the variable chain {*advisedBy(A, B)*, *publication(C, B)*, *publication(C, D)*, *professor(D)*}, it can be split into 4 literals, with the requirement that given an instantiation of A and B (or a ground atom built on *advisedBy*), $professor(D)$ will be set to *true* if there exists an instantiation of C and of D such that *publication(C, B)*, *publication(C, D)*, *professor(D)* are true. Therefore $professor(D)$ equal to True means that the entire chain is true. On the other hand, $publication(C, D)$ may be true with only $publication(C, B)$ true.

Starting from the set of variable chains that has been built, we build a set of variable literals, renaming variables when necessary. In order to achieve this, we use a tree-structure representation of the chains. The idea is that for each variable literal, there exists a single chain linking this literal to the target literal. The process is divided into three steps: switch, sort and featurize.

The switch operation looks for sequences of two variable literals in two different chains that would be similar up to a permutation of these two literals, as for instance $publication(C, A), publication(C, B)$ in one chain and $publication(C, B)$, $publication(C, A)$ in the other one.

Then chains are sorted as follows: a chain c_1 precedes a chain c_2 if c_1 is shorter than c_2 or c_1 has the same length as c_2 and l_1 precedes l_2 where l_1 and l_2 are respectively the first literals in c_1 and c_2 that differ. The order relation between literals corresponds to the alphabetical order of the name of their predicates, or to the (alphabetical or numerical) order of the first pair of variables that differs if the two literasl are based on the same predicate. We must underline that such an order relation is only introduced in order to sort chains and that is should be given no further meaning.

A tree structure is then built, processing variable chains in turn. During this operation, variables can be renamed, thus allowing to distinguish features. A mapping table is used, linking the old name to the new one. More precisely, given a variable chain $l_1, \ldots, l_p$, we first look for a prefix already existing in the

tree, possibly using a variable renaming as given in the mapping table. Let us call i the last index of this prefix. Then we search whether any literal l_j, with $j > i$ already occurs in the tree, which means that if this literal was introduced "as is" in the tree, this later would contain two similar nodes at different places, and thus with different meanings. If so, then two cases are considered to overcome this potential problem:

- l_j contains only variables of the target predicate: the chain l_j, ..., l_p is forgotten, since there exists another shorter chain linking l_j with the target predicate.
- at least one of its variables is not a variable of the target predicate. This variable is renamed, introducing a new variable. The link between this new variable and the original one is kept in the mapping table.

Finally, the chain is introduced in the tree, renaming variables according to the mapping table. Let us notice that once renaming is done, it is possible that the common prefix detected in the original chain is no longer a common prefix (due to the renaming of the variable) and then a new branch is created as needed.

We can now see how switching and sorting lead to a valuable organization of chains. Due to the sequential introduction of chains in the tree, chains that share a common prefix become neighbors, sorted so that there introduction in the tree is likely to generate as less branches as possible.

Example 3. Let us consider again the database given in Example 2. The following links are built:

1. {*[advisedBy student/1 1]*}
2. {*[advisedBy professor/2 1]*}
3. {*[advisedBy publication/1 2], [publication publication/1 1], [publication professor/2 1]*}
4. {*[advisedBy publication/1 2], [publication publication/1 1], [publication publication/2 2]*}
5. {*[advisedBy publication/2 2], [publication publication/1 1], [publication student/2 1]*}
6. {*[advisedBy publication/2 2], [publication publication/1 1], [publication publication/2 2]*}
7. {*[advisedBy publication/2 2], [publication publication/1 1], [publication professor/2 1]* }

Then variable chains are built. (In the current implementation, variables are represented by an integer; here, we write them X_i to improve lisibility.)

1. {*advisedBy(X1,X2), student(X1)*}
2. {*advisedBy(X1,X2), professor(X2)*}
3. {*advisedBy(X1,X2), publication(X3, X1), publication(X3,X2), professor(X2)*}
4. {*advisedBy(X1,X2), publication(X3,X1), publication(X3,X2), publication(X4, X2)*}

5. {*advisedBy(X1,X2) , publication(X3,X2), publication(X3,X1), student(X1)*}
6. {*advisedBy(X1,X2), publication(X3,X2), publication(X3,X1), publication(X4, X1)*}
7. {*advisedBy(X1,X2), publication(X3,X2), publication(X3,X4), professor(X4)*}

The first step switches some consecutive literals, in order to favor common prefixes. It is applied on the 5th and 6th clauses, thus leading to:

5'. {*advisedBy(X1,X2), publication(X3,X1), publication(X3,X2) , student(X1)*}
6'. {*advisedBy(X1,X2), publication(X3,X1), publication(X3,X2) , publication(X4, X1)*}

In the second step, they are sorted:

1. {*advisedBy(X1,X2), professor(X2)*}
2. {*advisedBy(X1,X2), student(X1)*}
3. {*advisedBy(X1,X2), publication(X3, X1), publication(X3,X2), professor(X2)*}
4. {*advisedBy(X1,X2), publication(X3,X1), publication(X3,X2), publication(X4, X2)*}
5. {*advisedBy(X1,X2), publication(X3,X1), publication(X3,X2), publication(X4, X1)*}
6. {*advisedBy(X1,X2) , publication(X3,X1) , publication(X3,X2) , student(X1)*}
7. {*advisedBy(X1,X2), publication(X3,X2), publication(X3,X4), professor(X4)*}

In the context of this example, sorting as a limited impact, but in larger datasets it usually has a much more important influence on the organization of links. Then the tree is built. Chains 1, 2 and 3 have only $advisedBy(X1, X2)$ as a common prefix. Chain 3 has a literal, namely $professor(X_2)$, that contains only variables occurring in the head, this literal is then removed. Chains 4 and 5 have a prefix of length 3 common to chain 3 and then differs. In chain 6, the last literal is removed. Finally, in Chain 7, variable X_3 is renamed in X_5 in order to distinguish the two occurrences in different situations of $publication(X_3, X_2)$. This leads to the tree:

advisedBy(X1,X2)
— *professor(X2)*
— *student(X1)*
— *publication(X3,X1)*
——– *publication(X3,X2)*
———— *publication(X4,X1)*
———— *publication(X4,X2)*
— *publication(X5,X2)*
——– *publication(X5,X4)*
———— *professor(X4)*

Once the set of features is built, we transform information in the database into a boolean table BT, where each column corresponds to a variable literal and each row corresponds to a true/false ground atom of the target predicate. Let

us assume that data concerning a given ground atom q_r is stored in row r. Let us also assume that column c corresponds to a given variable literal vl_c. There exists a chain vc of variable literals starting from the head literal and containing vl_c. $BT[r][c] = true$ means that there exists a ground chain gc_r starting from the ground atom q_r that makes vc true (or such that $vc \subseteq var(gc_r)$ up to a renaming of variables). Given a ground instance q_r, filling its row by considering all the ground chains starting from q_r is too expensive because it has to involve an exhaustive search in the database. We overcome this obstacle by inversely considering the variable chains. Each of them is then used to guide the search in the database. This search can be performed much faster using information about the order of predicates and positions of shared constants between two consecutive predicates in that chain. The notion of links allows us to filter the variable chains thus reducing the search.

3 Related Works

As previously mentioned, propositionalization approaches can be classified into approaches based on logic and approaches inspired from databases. The approaches inspired from databases, like Polka [2], RELAGGS [10] and RollUp [7] build attributes which summarize information stored in non-target tables by applying usual database aggregate functions such as count, min, max, ...

These works are quite different from the ones based on logic, which consider examples as first-order Herbrand interpretations and features (attributes) as conjunctions of first-order function-free atoms. The search then is based on a template (a set of ground atoms of which all arguments fall in exactly one of three categories: input, output, or constant) or mode declarations (define the predicates and assign a type and mode to each argument of these predicates). Our method belongs to the second trend based on logic, and is compared to these approaches.

Logic-based approaches can be divided into two families : the first one as for instance [9, 5] starts from information on the predicates (usually the modes), build features and then uses the database to filter relevant ones, the second one as for instance [14] or our work starts from the database, build ground conjunctions of atoms and variabilize them to get features. Let us illustrate these two kinds of approaches.

In [5], the authors introduce the notion of templates and features. A *template* is a conjunction of ground atoms, the arguments of which are either defined as inputs (+) or as outputs(-). To be a template, an atom must have at most one input argument and there exists a partial irreflexive order on the terms occurring in it ($c < c'$ if c and c' occur in the same atom, c as input and c' as output. From a template, *conjunctions of connected variable literals* can be built, under the conditions that variables can be instantiated in such a way that the ground atoms belong to the template. A variable can be positive, when it has exactly one input occurrence and no output occurrence, negative when it has no input occurrence and exactly one output occurrence, or neutral when it has at least one

input occurrence and exactly one output occurrence. A *feature* is then a conjunction of connected variable literals where all variables are neutral. Intuitively it means, that each variable must be introduced as output of a single atom and can then be described by several atoms where it occurs as input. Features are built by aggregating blocks, where blocks are conjunctions of variable atoms, either containing exactly one positive variable (positive block) or containing exactly one negative variable (negative block).

In [14], conjunctions of literals are also built from the database. They use mode declarations and they distinguish two types of predicates: path predicates with at least one output argument and check predicates with only input arguments. The features (called properties in their approach) that they build must contain at least a check predicate. It means that "pure" relational features, as for instance expressing that two persons are linked if they share a common publication ({*advisedBy(A,B), publication(C,A), publication(C,B)*} cannot be built. The check predicates play an important role in the search strategy, since given a saturated ground clause, they start from the atoms built with a check literal and look for the path allowing to connect them to the head.

Our method differs from the classical logic-based approaches both in the *semantic of the boolean table* and in the *search for features*.

Meaning of the table: To form an attribute-value table, most methods define each propositional feature (column) corresponding to a variable literal (SINUS and DINUS [8] for example) or to a conjunction of several variable literals (the genetic method [15], RSD [9], HiFi [4] and RELF [5]). In LBP, each propositional feature (column) corresponds to a variable literal and *each row corresponds to a true/false ground atom of the query predicate*.

Searching: To construct features, most methods use syntactical constraints in the form of template or mode declarations for limiting the search space, then apply some techniques to calculate the truth values for each feature. For example, the RELF system uses a block-wise technique to construct a set of tree-like conjunctive relational features while the others, like HIFI or RSD use the traditional level-wise approaches. The search in LBP does not rely on template or mode declarations, but on a synthetic representation of the dataset, namely the links of the chains, which allows building features as well as constructing the boolean table based on the regularities of these chains. The notion of chain is related to relational path-finding [12] and relational cliché [13].

4 Experiments

4.1 Systems, Databases and Methodology

We propose to evaluate LBP according to classification accuracy, as traditionally used in propositionalization[4, 5, 7, 9]. Accuracy is relevant as it expresses the ability of LBP to produce discriminative features. More information is given in the form of the F1 score of both positive and negative groups.

We compared LBP to two state-of-the-art logic-based systems: RELF [5] and RSD [9]. For the comparison of logic-based and database-inspired methods, we refer to [6, 7, 16] for further reading. We performed experiments on three popular datasets:

- *IMDB* consists of a database on films (6 predicates, 302 constants, 1224 true/false ground atoms). We learned the predicate *workedUnder* (i.e. who worked under the direction of who).
- *UW-CSE* describes an academic department (15 predicates, 1323 constants, 2673 ground atoms). We learned the predicate *advisedBy* (i.e., who is the advisor of who).
- *CORA* consists of citations of computer science papers (10 predicates, 3079 constants, 70367 true/false ground atoms). We learned the predicate *sameBib* (i.e. do two citations refer to the same paper).

LBP has first been built over the Alchemy platform [1], since as written in the introduction, the idea of a linked-based representation of the database had first been introduced for learning Markov Logic networks [11]. The datasets have thus been used in their Alchemy form. Each set consists of 5 folds, which have been used for cross-validation. A new version of LBP, independent from Alchemy, has been built in Java and this is this new version that is used in the following experiments.

To evaluate the outputs of LBP, RELF and RSD, the set of features that have been produced and their corresponding boolean tables have then been given as inputs of a discriminative tool. The three systems produce output data in the Weka format, and we have chosen to test the discriminative power of the features on decision tree classifiers, using the WEKA [17] implementation of J48 and REPTree. We have chosen these two decision tree learners as they differ on the way trees are built: J48 implements the C4.5 algorithm while REPTree is a faster tree decision learner.

In most of cases, datasets are highly unbalanced. Given the closed world assumption, many negative examples could be generated. Therefore beside the pre-existing negative ground atoms of the target predicate, additional negative examples are generated randomly, to reach a rate of 4 negatives per positive. We mean, that based on a closed world assumption, we consider that all positive examples are explicitely given and that negative examples can be deduced from these latter. In the considered datasets, no or few negative examples are explicitely given. We thus keep these explicit negative examples and generate a subset of the implicit ones. This is empiric but it seems to be a fair way to get enough negative examples while not leading to too much overhead.

4.2 Dataset formats

The data input formats of the three systems we compared do differ and choices have to be made to encode the requirements of the systems. We briefly detail

[1] http://alchemy.cs.washington.edu/

how we proceeded to adapt them, starting from the Alchemy-like format. LBP requires only a database expressed by a set of ground atoms, positive and negative examples; type information can be given to improve the efficiency of the system. On the other hand, RELF and RSD need more information.

Alchemy-like data used in LBP consists of a ".db" file that contains the ground atoms. It usually comes with a ".mln" file that contains both the description of the predicates and some elements to help building a Markov logic network (which is out of the scope of this paper). The target predicate is defined at runtime. The description of the predicates can be very simple. In our case we only consider their name and arity, together with the type of arguments (e.g. *advisedBy(person, person)*). No mode or other additional declaration bias is used.

RELF learns by interpretation, which means that its input data consists of a set of independent world interpretations (i.e. a set of ground facts), which are annotated as valid or not. To comply to this data organization, we proceed as follows. Starting from each ground atom of the target predicate, we build an interpretation based on the saturation of this ground atom. We tag this interpretation as true of false based on the sign of the target atom. This latter is of course removed from the interpretation. In the case where the arity of the target predicate is higher that 1, as for instance *advisedBy*, we have to specify in the interpretation the constants occurring in the target atom. We thus introduce an additional predicate that is similar to the target one, but which is always true. Beside this data file, Relf needs an additional file that contains a template to guide its data exploration. Let us insist on the fact that this template has a high influence on the results.

RSD data input can take several forms. In the approach we use, input data consists of three files. First a ".b" knowledge base contains the mode declaration of the predicates that distinguishes between the target predicate (modeh) and the body predicates (modeb). The knowledge base also contains the positive ground atoms of the body predicates. Two other files are needed, a ".f" one and a ".n" one, that respectively contain the true and false ground atoms of the target predicate. The arity of the target predicate must be equal to 1. Thus, we have had to modify our datasets by introducing additional predicates: a new target predicate of arity 1, the variable of which is a new one, and *linking* predicates of arity 2 that link the new variable to the ones of the original target predicate. The new variable is set as an input one in the modes declaration.

We have used similar feature declaration biases for RSD and RELF. For LBP, we arbitrarily set the maximal length of considered g-chains (v-chains) to $k = 4$, in order to explore a rich while tractable search space. For RSD, due to the additional predicates we introduced, we set this maximum length to 6.

Beside dataset formats, we must also notice that LBP and RELF are able to take both learning and test datasets as input and produce the respective output files, while RSD is not. We thus adapted the tests as follows: with LBP and RELF we conducted a 5-fold cross validation based on our original folds, while with

RSD we merged all of the folds as a single input, thus getting a single output and letting Weka process with its built-in, 10-fold, cross validation.

4.3 Results

We present one table per dataset, containing for each system the global accuracy and the F1scores for positive examples (F+) and negative examples (F-). We used two versions of LBP, one that learns features based on the positive target atoms only (LBP+), and one that learns features based on both positive and negative target atoms(LBP-). Tables 1, 2 and 3 respectively correspond to the experiments with IMDB, UW-CSE and CORA. The presented results correspond to the average results of the 5- or 10-fold cross validation process. On each line the best average value if set in bold face.

		LBP+	LBP-	Relf	RSD
J48	Accuracy	**97.6**	92.4	92.8	84.6
	F+	**0.95**	0.85	0.86	0.90
	F-	**0.98**	0.946	0.95	0.69
REPTree	Accuracy	**97.6**	92.6	92.8	84.9
	F+	**0.95**	0.82	0.86	0.90
	F-	**0.98**	0.9	0.95	0.70

Table 1. Imdb experiments

		LBP+	LBP-	Relf	RSD
J48	Accuracy	90.4	**93.9**	91.1	85.8
	F+	0.79	0.86	0.81	**0.91**
	F-	0.94	**0.96**	0.94	0.71
REPTree	Accuracy	91.6	**94.5**	91.3	85.8
	F+	0.82	0.87	0.82	**0.91**
	F-	0.95	**0.96**	0.94	0.72

Table 2. Uw-cse experiments

We have got no results with RSD and REFL on Cora. More precisely, the systems terminate but provide no results. We can observe that in general the best accuracy is achieved with one of the two versions of LBP. Nevertheless, due to the fact that, depending on the dataset, either one or the other performs better, we cannot conclude that one of them is globally more performant from a statistical significance point of view.

		LBP+	LBP-	Relf	RSD
J48	Accuracy	87.9	86.8	-	-
	F+	0.76	0.74	-	-
	F-	0.92	0.91	-	-
REPTree	Accuracy	87.2	86.9	-	-
	F+	0.75	0.74	-	-
	F-	0.91	0.91	-	-

Table 3. Cora experiments

We also achieve the best F1Scores, except on UWCSE, where RSD performs better on the positive F1Score. Depending on the dataset, the decision tree learning algorithm might have some influence (UWCSE) or nearly no influence (IMDB) on the results at the average level. Nevertheless, even in this second case, differences might be noticed when considering some folds.

The important point is that these satisfying performances are obtained with a method that introduces no learning bias, except the types of variables, which is much lighter than the biases of REFL and RSD.

Considering time, LBP is the slowest system (about twice than the two other systems for UWCSE and IMDB). Implementation considerations might explain it partially, but on the large datasets, the fact that no declaration biases, such as modes, are available, makes the dataset exploration much longer.

Considering features, on UW-CSE, LBP+ produces c.a. 300 features, LBP- c.a. 550 features, RELF c.a. 130 features and RSD c.a. 300 features. On IMDB, LBP+ produces c.a. 30 features, LBP- c.a. 40 features, RELF c.a. 10 features and RSD c.a. 100 features. The fact that we produce much more features than RELF can be explained by at least two reasons. First, we produce features that consist of a single variable literal. We thus have several features when other systems produce a single conjunction. Second, due to the tree structure of our graph of features, we have a lot of features that are set and kept "just to" materialize a path to a leaf feature. Nevertheless, we surprisingly produce less features than RSD.

Based on these results, we can conclude that our system is competitive to the state-of-the-art propositional systems on these three benchmark datasets.

5 Conclusion and Future Work

In this paper, we introduce a linked-based representation of the database allowing to capture relational structures in the database, and we give a first way of integrating it in a propositional learner. Our main contribution is mainly on this linked-based representation allowing to learn features with nearly no information (except types of predicates). Another original contribution is the idea of splitting features into literals, relying on the fact that they form a chain.

Further works can be done on this representation. In the system that we have developed, learned features are split into ground atoms. Such features could also be used as such as in traditional propositional learners.

Although the system was designed for avoiding the user to give biases, modes could easily be added, thus allowing to reduce the number of links. On the other hand, most logical-based propositional learners need information: at least type and mode declarations for predicates, more sophisticated information, as for instance templates, which allows to reduce the search space. Giving such templates is not so easy. Our linked-based representation could perhaps be used as a preliminary step to learn templates.

References

1. Alphonse, É., Rouveirol, C.: Selective propositionalization for relational learning. In: PKDD'99. Volume 1704 of LNCS, Springer (1999) 271–276
2. Knobbe, A.J., de Haas, M., Siebes, A.: Propositionalisation and aggregates. In: PKDD'01. Volume 2168 of LNCS, Springer (2001) 277–288
3. De Raedt, L.: Logical and Relational Learning. Springer (2008)
4. Kuželka, O., Železný, F.: Hifi: Tractable propositionalization through hierarchical feature construction. In: Late Breaking Papers, ILP'08. (2008)
5. Kuželka, O., Železný, F.: Block-wise construction of tree-like relational features with monotone reducibility and redundancy. Mach. Learn. **83**(2) (2011) 163–192
6. Krogel, M.A., Rawles, S., Železný, F., Flach, P.A., Lavrac, N., Wrobel, S.: Comparative evaluation of approaches to propositionalization. In: ILP'03. Volume 2835 of LNCS, Springer (2003) 197–214
7. Lesbegueries, J., Lachiche, N., Braud, A.: A propositionalisation that preserves more continuous attribute domains. In: ILP'09. (2009)
8. Lavrac, N., Dzeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood (1994)
9. Lavrac, N., Zelezný, F., Flach, P.A.: Rsd: Relational subgroup discovery through first-order feature construction. In: ILP'02. Volume 2583 of LNCS, Springer (2002) 149–165
10. Krogel, M.A., Wrobel, S.: Transformation-based learning using multirelational aggregation. In: ILP'01. Volume 2157 of ILP, Springer (2001) 142–155
11. Dinh, Q.T., Exbrayat, M., Vrain, C.: Discriminative markov logic network structure learning based on propositionalization and chi^2-test. In: ADMA'10. Volume 6440 of LNCS, Springer (2010) 24–35
12. Richards, B.L., Mooney, R.J.: Learning relations by pathfinding. In: AAAI'92, AAAI Press / The MIT Press (1992) 50–55
13. Silverstein, G., Pazzani, M.J.: Relational clichés: Constraining induction during relational learning. In: ML'91, Morgan Kaufmann (1991) 203–207
14. Motoyama, J., Urazawa, S., Nakano, T., Inuzuka, N.: A mining algorithm using property items extracted from sampled examples. In: ILP. (2006) 335–350
15. Braud, A., Vrain, C.: A genetic algorithm for propositionalization. In: ILP. (2001) 27–40
16. Kuzelka, O., Zelezný, F.: Block-wise construction of acyclic relational features with monotone irreducibility and relevancy properties. In: ICML'09, ACM (2009) 72
17. Machine Learning Group at University of Waikato: Data mining software in java. `http://www.cs.waikato.ac.nz/ml/weka/`

MicroRNAs Analysis by Hypothesis Finding Technics

Andrei Doncescu[1,2], Katsumi Inoue[1], and Anne Pradine[2]

[1] National Institute of Informatics Japan
[2] Cancer Institute of Toulouse, France

Abstract. The cell is an entity composed of several thousand types of interacting proteins. Our goal is to comprehend the cancer regulation mechanisms using the microRNAs. MicroRNAs are present in almost all genetic regulatory networks acting as inhibitors targeting mRNAs. In this paper, it is shown how the Artificial Intelligence description method functioning on the basis of Inductive Logic Programming can be used successfully to describe essential aspects of cancer mechanisms. The results obtained show new microRNAs markers for melanoma metastasis.

1 Introduction

New technologies have been developed developed to measure the expression level of thousands of genes simultaneously. These genomic-scale snapshots of gene expression (i.e. how much each gene is "turned on") are creating a revolution in in biology. Genes encode proteins, some of which in turn regulate other genes. Understanding genetic and metabolic networks is of the utmost importance. These networks control essential cellular processes and the production of important metabolites in microorganisms, and modeling such networks from model organisms will drive applications to other less characterized organisms, which have a high biotechnological potential.

The study of signaling events appears to be a key to the research, biological, pharmacological and medical. The spread of these types of signals are not changing the behavior of proteins on three levels: regulation of the activity, interaction and expression. The three levels are synchronized in a strong momentum that leads to changes in protein activity. Since a decade signaling networks have been studied using analytical methods based on the recognition of proteins by specific antibodies. Parallel DNA chips (microarrays) are widely used to study the co-expression of candidate genes to explain the etiology of certain diseases, including cancer.

From the standpoint of Artificial Intelligence cells are sources of information that include a myriad of intra and extra cellular signals that as the ultimate goal of optimal output describing metabolic proteins. Diseases and cancer in particular can be seen as a pathological alteration in the signaling networks of the cell.

The study of gene networks poses problems identified and studied in Artificial Intelligence over the last twenty years. It is clear identified that the reasoning have to handle incomplete, uncertain, revisable, contradictory and multiple sources of information. The logical representation of signaling pathways is not complete: biological experiments provide a number of protein interactions but certainly not all. On the other hand the conditions and difficulty of some experiments lead to obtain data which are not always accurate. Some data may be very wrong and must be corrected or revised in the future. Finally the information coming from different sources and experiences and can be contradictory. From this observation, generally for most real human activities, it must, however, reason and make decisions. It is the goal of logics of uncertainty and in particular of nonmonotonic logics.

The logical approach provides an intuitive method to provide explanations based on the expressivity of relational language. For example, logic can represent biological networks such as gene regulation, signaling transduction, and metabolic pathways. Unlike other approaches, this method lets us introduce background theory, observations and hypotheses within a common declarative language. It also provides the basis for the three main forms of inference, i.e., deduction (prediction), abduction (explanation) and induction (generalization).

In a quarter of a century of automatic demonstration, propositional calculus has been largely neglected in favor of more substantial logics, such as predicate calculus or certain kind of non-classical logic. In the case of propositional calculus, the algorithms of automatic demonstration, be they of syntactical or semantic nature, are characterized by their great simplicity. It is thus easy to analyze their limits and weak point, and to work towards an improvement in their performance.

Another fundamental asset of propositional calculus is its decidability : we can implement algorithms which will give a result within a finite and reasonable time, even for non-trivial examples. In first-order logic, such algorithms cannot exist.

1.1 Causality and classical inference

If the inference of classical logic $A \rightarrow B$ or $A \dashv B$ is fully described formally, with all the "good" logic properties (tautology, not contradiction, transitivity, contraposition, modus ponens, ...), a description of the properties of causality is less simple. Causality can not be seen as a classical logic relation. In this paper, it is described and use a very simple form of causality necessary and probably sufficient for the application to the cell.

To provide the causal links between our relations cause and blocks in a classical language (propositional calculus or first order logic) it is necessary to do two things :

1. Describe the internal characteristics of relations and causes
2. Describe the links between these relations and classical logic

Taking into account the aspect of discovery (abduction, field production) the hypothesis theory is applied. The default logic raises a number of theoretical and practical problems. To solve these problems we will use the Hypotheses Theory, wich is a nonmonotonic logic. A fragment of this logic is close to stable model. One advantage of hypotheses theory is that the information is described in a classic bimodal logic. It will then be possible to use this formalism to describe the set of all information, uncertain or not. This logic will also allow the use a consequence finding algorithms to treat abduction.

2 Integrating induction and abduction in CF-induction

Both *induction* and *abduction* are ampliative reasoning, and agree with the logic to seek hypotheses which account for given observations or examples. That is, given a background theory B and observations (or positive examples) E, the task of induction and abduction is common in finding a hypothesis H such that

$$B \wedge H \models E, \tag{1}$$

where $B \wedge H$ is consistent. There are several discussions on the difference between abduction and induction in the philosophical and pragmatic levels. On the computational side, induction usually involves *generalization*, while abduction gives *minimal explanations* for individual observation.

Inverse entailment (IE) is a logically principled way to compute abductive and inductive hypotheses H in (1) based on the logically equivalent transformation of the equation (1) to

$$B \wedge \neg E \models \neg H. \tag{2}$$

The equation (2) says that, given B and E, any hypothesis H *deductively* follows from $B \wedge \neg E$ in its negated form. The equation (2) is seen in literature, e.g., [12] for abduction and [15] for induction. The equation (2) is useful for computing abductive explanations of observations in abduction. This is because, without loss of generality, in abduction E is written as a ground atom, and each H is usually assumed to be a conjunction of literals. These conditions make abductive computation relatively easy, and *consequence-finding* algorithms [12] can be directly applied.

In induction, however, E can be clauses and H is usually a general rule. Universally quantified rules for H cannot be easily obtained from the negation of consequences of $B \wedge \neg E$. Then, Muggleton [15] introduced a *bridge formula* U between $B \wedge \neg E$ and $\neg H$:

$$B \wedge \neg E \models U, \quad U \models \neg H.$$

As such a bridge formula U, Muggleton considers the conjunction of all unit clauses that are entailed by $B \wedge \neg E$. In this case, $\neg U$ is a clause called the *bottom clause* $\bot(B, E)$. A hypothesis H is then constructed by generalizing a sub-clause of $\bot(B, E)$, i.e., $H \models \bot(B, E)$. This method with $\bot(B, E)$ is adopted

in Progol, but it has turned out that it is incomplete for finding hypotheses satisfying (1).

In [13], Inoue proposed a simple, yet powerful method to handle inverse entailment (2) for computing inductive hypotheses. The resulting method called *CF-induction* does not restrict the bridge formula U as the set of literals entailed by $B \wedge \neg E$, but consider the *characteristic clauses* [12] of $B \wedge \neg E$, which obviously generalizes the method of the bottom clause. CF-induction then realizes sound and complete hypothesis finding from *full clausal theories*, and not only definite clauses but also non-Horn clauses and integrity constraints can be constructed as H.

In most previous inductive methods including Progol [15], there are syntactical restrictions such that: (i) each constructed hypothesis in H is usually assumed to be a single Horn clause, (ii) an example E is given as a single Horn clause, and (iii) a background theory B is a set of Horn or definite clauses. From the viewpoint of applications, these restrictions are due to the easiness for handling such formulas. An extension to multiple non-Horn clauses in B, E, and H is, however, useful in many applications.

First, an extension allowing *multiple clauses* in either a hypothesis H or an observation E is essential in applications of abduction. In fact, recent work on abductive inference in metabolic pathways [16] uses an independent abductive procedure to obtain a set of literals that explain an observation. In general, there are multiple missing data to account for an observation. This abductive inference is independently computed in [16] not by Progol, and the inductive process for generalization takes place by Progol only after abductive hypotheses have been obtained. On the other hand, CF-induction can be used to compute abductive explanations simply by taking the bridge formula U as a deduced clause. CF-induction thus integrates induction and abduction from the viewpoint of inverse entailment through consequence-finding [13].

Second, an extension to *non-Horn clauses* in representation of B, E, H is also useful in many applications. For example, indefinite statements can be represented by disjunctions with more than one positive literals, and integrity constraints are usually represented as negative clauses. The clausal form is also useful to represent causality. For example, when we want to represent *inhibition* of reaction in a causal pathway network, positive and negative literals with the predicate like `inhibited` can be used in the premise of each causal rule, which results in a non-Horn clause (we will see an example afterwards). Again, the inductive machinery of CF-induction can handle all such extended classes.

Third, introducing *multiple, non-Horn clauses* in a hypothesis H is an unifying extension that combines the first and second extensions. In this case, a hypothesis H forms a *theory*, which can also account for multiple observed data at once.

In our application to cancer analysis, given the background theory of network structures of a pathway and observations, we need a hypothesis H that explains the behavior of the metabolic system. In principle, such a hypothesis consists

of multiple non-Horn clauses each of which represents a causal relation. CF-induction is thus particularly useful for this type of applications.

Here the structure of representation is based on the specification of CF-induction program, which is compatible with the consequence-finding program SOLAR [19] and the TPTP format for theorem proving. SOLAR is a Java implementation of the tableaux variant of SOL resolution [12].

For example, the input clauses can be described as

```
input_clause(axiom1,    bg, [-p(X), -q(X), r(X)]).
input_clause(example1, obs, [r(a)]).
production_field([predicates(pos_all), length < 3]).
```

Here, `axiom1` and `example1` are ID names of clauses, and `bg` and `obs` represent background knowledge and observation, respectively. The axiom1 means $\rceil p(x) \vee \rceil q(x) \vee r(x)$. Each clause is represented as a list of literals. The predicate `production_field` indicates the *production field* of SOLAR, and this example allows it to generate consequences consisting of less than 2 positive literals. In this way, a production field can be used to specify an *inductive bias* in CF-induction. There is other meta information to control deduction in SOLAR such as the search strategy and the depth limit.In this case, CF-induction produces the abductive hypothesis:

```
Hypotheses: [ [p(a)], [q(a)] ]
```

The current CF-induction program has several *generalizers*, which, given a set T of clauses, produce a set S of clauses such that $S \models T$. These basic generalizers include *anti-instantiation*, *reverse Skolemization*, *Plotkin's least generalization*, and *dropping literals* (see [13]).

Therefore, CF-induction is related to top-down decision tree learning algorithm generating a set of rules in the form of predicate logic clauses which can be used to separate the classes.

2.1 Signaling Pathway Representation

In this paper, it is used only a propositional representation. In practice the detailed study of interactions will be asked to represent increases or decrease some biological quantities which could be protein concentration. It therefore falls outside the scope of propositional but the basic problems are the same. To represent a change in concentration is for example possible by using predicates such as "increased" or "decrease" [3].

To describe interactions between genes/proteins we use a language L of classical logic (propositional). The proposition A (resp. $\neg A$) says that A is true (false). We are in a logical framework, so it is possible to represent almost everything in a natural way. Interactions between genes/proteins is a very simple form of causality.

3 MicroRNAs

MicroRNAs are small RNA molecules that were discovered in the 1990s in animals and plants, and which play an essential role in controlling gene expression. In human being, more than 500 microRNAs have been identified, and we now know that their dysfunction is associated with several diseases, including cancer. The microRNAS play an important role of not specific inhibition in many circumstances of the cell life, like chromatin clock control and have a big influence on many metabolic controls of functions like energy systems, cell cycle and defense systems against pathogens.

MicroRNAs are present in almost all genetic regulatory networks acting as inhibitors targeting mRNAs, by hybridizing at most one of their triplets, hence acting as translation factors by preventing the protein elongation in the ribosome. MicroRNAs act as source nodes in the interaction graph of genetic regulatory networks, which are made of elements, the genes, in interaction through the protein they express and control important cell or tissue functions like proliferation, differentiation, energy systems maintenance, and more generally homeostasis [3,4].

3.1 MicroRNAs Identification in Melanoma

The microRNAs appear increasingly as crucial actors in oncogenesis (miR-21 in breast cancer) or as a tumor suppressor. Furthermore, the expression profiles of microRNAs in solid tumors of different origins have been made using prognostic values.

In 2008 the circulating microRNA were revealed for the first time in serum and plasma, with very different profiles between healthy donors (which have a similar expression profile) and patients with breast cancer, lung , prostate with specific expression patterns. In addition, these microRNAs have a high stability since they form complexes with lipids or lipoproteins, allowing the resistance to the activity of RNase and DNase. They can also withstand harsh experimental conditions such as high temperature or high pH variation. These data therefore show the circulating microRNAs (noninvasive) could be novel plasma biomarkers in oncology.

Melanoma is a cancer of the skin or mucous membranes, developed at the expense of melanocytes. In most cases, it develops first on the skin but it is common to find melanoma of the eye (choroidal melanoma), mucous membranes (mouth, anal canal, vagina), or even more rarely internal organs. The incidence of this disease is increasing worldwide. In France, it was evaluated in 2010 at approximately 7-9 new cases per year per 100 000 individuals with a mortality rate ranging from 1.2 to 1.5 individuals.

For this study, we collected 29 subjects having metastatic melanomas and 5 healthy volunteers. The micro-arrays are spotted with microRNA of human, murine, and viral control. In all, there were 4608 different microRNA spots and 2000 different microRNA.

In order to highlight the existence of circulating microRNA biomarkers in melanoma, a technique was developed for extracting microRNAs from the plasma of healthy donors. This lets us collect a larger amount purified RNA from the serum). The expression profile of microRNA was obtained using by a technique of monochrome chip. These chips were spotted (labeled with fluorescent HY3) with probes modified with bases LNA (Locked Nucleic Acid home Exiqon). The goal was to optimize the specificity and sensitivity of probes (annealing temperature adapted to detect microRNA with a low percentage of GC). Each microRNA was represented by two different spots. These chips were analyzed using image analysis software GenePix Pro 6.1.0.4 (Axon Instruments).

3.2 MicroRNAs expressed between 2 populations : healthy subjects and patients with melanoma

Using Bayesian approach (Limma) [17] , the original human microRNA differentially expressed between the two groups of subjects were identified (Fig.1).

Name	logFC	AveExpr	t	P.Value	adj.P.Val	Log-odds
hsa-miR-323-5p/mmu-miR-323-5p/rno-miR-323*	-0,48376	1,41123	-12,52041	0,00000	0,00164	4,83161
hsa-miR-1290	-0,08370	1,87079	-4,86814	0,00002	0,01032	2,35283
hsa-miR-1283	0,27753	1,44350	8,80186	0,00005	0,01768	2,31022
hsa-miR-302a*	-0,48901	1,43369	-9,76053	0,00018	0,03215	0,96393
hsa-miR-299-3p	0,05898	1,66613	4,41086	0,00008	0,02171	0,95292
hsa-miR-191*	0,15563	1,46111	5,31796	0,00033	0,04484	0,60907
hsa-miR-1246	-0,06650	1,85606	-4,16537	0,00016	0,03215	0,22045
hsa-miR-766	0,05295	1,69399	4,11137	0,00020	0,03215	0,05787
hsa-miRPlus-F1221	0,05284	1,61874	3,91631	0,00035	0,04484	-0,52358

Fig. 1. The different microRNAs expressed between the 2 populations (healthy subjects and patients with melanoma).

4 Discretization of Continous Values

Discretizing time series is a research domain on its own and many works [6, 23] have been conducted recently. Our practical problem is that we want to have a statistically relevant (unsuppervised) discretization for N chemical compounds concentrations over time. For that purpose, we compute an appropriate number of levels (5) in regard to a Bayesian score such as Bayesian Information Criterion [24].

We use continuous (Gaussian) hidden Markov models with parameter tying, which means that each chemical compound has a corresponding HMM but all the N Gaussian HMM share the same parameters (means and covariances), to share the same discrete outputed levels between the different compounds of one experiment. This relevant discretized levels of concentration are computed through expectation maximisation with maximum a posteriori [3, 7] . The level of microRNA expression are represented as *: very low, low, medium, high, very high*

4.1 Analysis of microRNAs expressed in melanoma

The highly aggressive character of melanoma makes it an excellent model to probe the mechanisms underlying metastasis, the process by which cancer cells travel from the primary tumor to distant sites in the body. Therefore, the goal of this analysis is to find out a microRNA signature in the case of aggressive melanoma. This signature is represented by a logical clause including the relevant microRNA, age and Breslow index. One hundred thirsty microRNAs have been identified and have been selected for modelling but only 51 has at least five valid replicas. After identifying 23 microRNAs, which already have extensively been studied a file containing the among of these microRNA compared with the healthy persons was created. The 23 microRNA candidates to melonama signature are : *hsa-let-7b, hsa-miR-17, hsa-miR-18a, hsa-miR-20a, hsa-miR-21, hsa-miR-34a, hsa-miR-130a, hsa-miR-141, hsa-miR-143, hsa-miR-145, hsa-miR-146a, hsa-miR-152, hsa-miR-155, hsa-miR-185, hsa-miR-191, hsa-miR-200c, hsa-miR-221, hsa-miR-222, hsa-miR-338-3p, hsa-miR-1246, hsa-miR-1290, hsa-miR-2110, miR-27b.*

The representation contains information about microRNAs (23), Breslow index, age, relapse and metastasis or die.

A logical predicate *expressed* for each patient has been defined. Each patient is characterized by 23 microRNAs. An important aspect of the representation is to consider the clause as background in CF-induction (bg).

```
Example1: [input_clause(patient11,bg,[hsamiR20a(patient1,medium),
hsamiR21(patient1,medium),
...
hsamiR34a(patient1,low),hsamiR222(patient1,medium),hsamiR3383p(patient1,low)]). ]
```

Concerning the representation of Breslow Index we decided to introduce a predicate *breslow*

```
input_clause(patient17, obs, [breslow(patient1,medium)]).
```

The age of patients has been discretized according with the next empirical information provided by doctors (low: 1-30, medium: 31-60 and high: 61 and more) and the clause is similar to Breslow index. Relapse is an important clinical parameter related to the number of days that the patient kept his "health". It is given as low (1-500 days), medium (501-1000 days) and high (more of 1001 days).

The predicate *relapsed(X, Y)* contains 2 atoms : the first one is the patient and the second parameter gives information about the number of days considered "healthy".

The expert knowledge is represented by the microRNA well known in melanoma development or invasive cancer.

```
input_clause(bg1, bg, [-hsamiR20a(X,high),
-hsamiR21(X,high), ...
-hsmaiR34a(X,low),
relapsed(X,high)]).
```

Basically, melanomas, expressions of miRNA-20a, miRNA-106a, miRNA-17, miRNA-21, and miRNA-34a are significantly up-regulated, while miRNA-145 and miRNA-204 expression are significantly down-regulated. In our case we have considered the microRNA-34a down-regulated. microRNA-34a maps to a chromosome 1p36 region that is commonly deleted and it has been found to act as a tumor suppressor through targeting of numerous genes associated with cell proliferation and apoptosis. The patients analyzed have advanced ages and our statistical analysis showed a low expression in this case.

4.2 Results in microRNA signature

The goal of our logical model is to identify the patients who have a fast evolution of cancer. We have used CF-Induction to obtain diagnosis rules based on these predicates. CF-Induction searches for a set of st-order predicate logic clauses that distinguish between two classes, one which is presented to the learner as positive and negative examples. CF-Induction is independent of the ordering of the positive examples. Another important aspect is the possibility to check whether B and H are consistent or not.

The most noteworthy result is the discover of a signature in the case of aggressive melanoma represented by 5 microRNAs : 20a,21,34a, 222 and 333-3p. This results has been obtained from the analysis of the number of examples explained by these microRNA and compared with the the results published by PubMed. The last two microRNAs are not well-known having a strong oncogene or anti-oncogene regulation.

The idea was to produce all predicates : beginquote

```
production_field(
[
predicates(all)
]
).
```

```
[relapsed(patient25, low), -hsamiR20a(patient25, medium)]
```

```
[relapsed(patient25, low), -hsamiR21(patient25, medium)]
[relapsed(patient25, low), -hsamiR34a(patient25, low)]
[relapsed(patient25, low), -hsamiR222(patient25, medium)]
[relapsed(patient25, low), -hsamiR3383p(patient25, low)]
```

Patient 25 was born in 1947 and died in 2011. The screening was done during stage iV of his cancer.

Using CF-induction with the production field :

```
production_field(
  [
    length <= 8,
    term_depth < 3,
  predicates([hsamiR222(_,_),hsmaiR3383p(_,_),age(_,_),relapsed(_,_)])
  ]
  ).
```

we have obtained the next result.

```
B & H is consistent

Hypotheses:
[hsamiR21(patient25, low), hsamiR3383p(patient25, low),
hsamiR222(patient25, verylow), hsamiR20a(patient25, low),
-age(patient25, high)]
```

This means the patient's age explains the observation of 4 microRNA: microRNA 21,20a, 222 and 338-3p. Therefore if we corroborate the 2 results, the microRNA 34a is a potential marker of melanoma in the case of older people. Basically microRNA 34a, maps to a chromosome 1p36 region that is commonly deleted and it has been found to act as a tumor suppressor through targeting of numerous genes associated with cell proliferation and apoptosis. A similar result has been obtained for the patient 24, who is born in 1932 and he is still living. Her screening was done during stage III of her cancer. It seems a strong correlation may exist between age and the microRNA 34a.

Another interesting result concerns patient 17, who has a very low concentration of microRNA 338-3p. At first glance, this could be considered a cancer marker, but the next results :

```
[relapsed(patient17, high), -age(patient17, medium)]
[hsamiR3383p(patient17, veryverylow), -age(patient17, medium)]
```

show that in this case age is the most important parameter of melanoma evolution. This person was born in 1962; the melanoma was identified for the first time in 2006. 6 years later she remains in phase IV.

5 Conclusion

Understanding genetic and metabolic networks is of the utmost importance. With the development of DNA microarrays, it is possible to simultaneously analyze the expression of up to thousands of genes and to construct gene networks based on inferences over gene expression data. We have found in this study that 5 microRNAs : 20a,21,34a, 222 and 333-3p are an important indicator of melanoma evolution. The microRNA 333-3p is not known as significantly down-regulated in the case of aggressive melanoma. Another surprising result is the correlation between age and microRNA 34a. Therefore, we are confident that future work will let us appreciate the complexity of micro-RNAs in the case of the invasive melanoma and going further to find out a signature of homeostasis. Homeostasis, the subtle balance between proliferation, differentiation and cell death of an organism is a complex phenomenon still little understood. Researchers are pointing more and more the important role of micro-RNA in this phenomenon.

References

1. Bandiera, S., Rberg, S., Girard, M., Cagnard, N., Hanein, S., Chrtien, D., Munnich, A., Lyonnet, S., Henrion-Caude, A., 2011. Nuclear Outsourcing of RNA Interference Components to Human Mitochondria. PLoS ONE 6, e20746.
2. Barbarotto E, Schmittgen TD, Calin GA. MicroRNAs and cancer: profile, profile, profile. Int J Cancer 2008; 122 (5): 969-77.
3. Beal M. J.: Variational Algorithms for Approximate Bayesian Inference. PhD. Thesis, Gatsby Computational Neuroscience Unit, University College London (2003).
4. A. Doncescu,Y. Yamamoto, K. Inoue. Biological Systems Analysis using Inductive Logic Programming *The 2007 IEEE International Symposium on Bioinformatics and Life Science Computing BLSC07*, Niagara Fall, Ontario, Canada, 2007.
5. Cummins JM, Velculescu VE. Implications of micro-RNA profiling for cancer diagnosis. Oncogene 2006; 25 (46): 6220-7.
6. Geurts P.: Pattern extraction for time-series classification. Proceedings of PKDD 2001, 5th European Conference on Principles of Data Mining and Knowledge Discovery, LNAI 2168, 115-127 (2001).
7. Gauvain J.-L. and Lee C.-H.: Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains. IEEE Transactions on Speech and Audio Processing, Vol.2, Issue 2, pp.291-298 (1994).
8. Ji S., Krishnapuram B., and Carin L.: Variational Bayes for continuous hidden Markov models and its application to active learning. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.28, Issue 4, pp.522-532 (2006).
9. R. King, K. Whelan, F. Jones, P. Reiser, C. Bryant, S. Muggleton, D. Kell, and S. Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427:247–252, 2004.
10. Kitano H.: Systems Biology Toward System-level Understanding of Biological Systems Kitano. In Science Vol. 295. no. 5560, pp. 1662-1664 (2002).
11. K. Inoue: Induction as consequence finding. Machine Learning, 55:109–135, 2004.
12. K. Inoue. Linear resolution for consequence finding. *Artificial Intelligence*, 56:301–353, 1992.
13. K. Inoue. Induction as consequence finding. *Machine Learning*, 55:109–135, 2004.

14. R. J. Mooney Integrating abduction and induction in machine learning. In Working Notes of the IJCAI97 Workshop on Abduction and Induction in AI, 37–42 (1997).
15. S. Muggleton. Inverse entailment and Progol. *New Gen. Comput.*, 13:245–862, 1995.
16. A. Tamaddoni-Nezhad, R. Chaleil, A. Kakas and S. Muggleton. Application of abductive ILP to learning metabolic network inhibition from temporal data. Machine Learning, 64:209–230, 2006.
17. Smyth, G. K. (2005). Limma: linear models for microarray data. *Bioinformatics and Computational Biology Solutions using R and Bioconductor, R. Gentleman, V. Carey, S. Dudoit, R. Irizarry, W. Huber (eds.), Springer, New York*, pages 39720.
18. Nabeshima H., Iwanuma K., and Inoue K.: SOLAR: A Consequence Finding System for Advanced Reasoning. Proceedings of the 11th International Conference TABLEAUX 2003, Lecture Notes in Artificial Intelligence, Vol. 2796, pp. 257-263, Springer (2003).
19. H. Nabeshima, K. Iwanuma and K. Inoue. SOLAR: a consequence finding system for advanced reasoning. *Proc. Eleventh Int. Conf. Automated Reasoning with Analytic Tableaux and Related Methods*, *Proc. TABLEAUX 2003*, LNAI 2796, pages 257–263, Springer, 2003.
20. Schultz J, Lorenz P, Gross G, Ibrahim S, Kunz M. MicroRNA let-7b targets important cell cycle molecules in malignant melanoma cells and interferes with anchorage-independent growth. Cell Res 2008; 18 (5): 549-57.
21. Yamamoto Y., Inoue K.,Doncescu A. : ntegrating abduction and induction in biological inference using CF-induction Elements of Computational Systems Biology by Huma M. Lodhi and Stephen H. Muggleton Wiley Book
22. Y. Yamamoto, K. Inoue, A. Doncescu : "Abductive Reasoning in Cancer Therapy" IEEE AINA 2009, May 26-29, Bradford
23. Keogh, E. and Lin, J. and Fu, A.: HOT SAX: efficiently finding the most unusual time series subsequence. 5th IEEE International Conference on Data Mining (2005).
24. Schwarz G.: Estimating the dimension of a model. Annals of Statistics, Vol.6, No.2, pp.461-464 (1978).

A Problog Model For Analyzing Gene Regulatory Networks

António Gonçalves[1], Irene M. Ong[2], Jeffrey A. Lewis[3] and Vítor Santos Costa[1]

[1] Faculty of Sciences, Universidade do Porto
CRACS INESC-TEC and Department of Computer Science
Porto, Portugal 4169-007
Email: up100378013@alunos.dcc.fc.up.pt, vsc@dcc.fc.up.pt
[2] Great Lakes Bioenergy Research Center, University of Wisconsin
Madison, WI 53706
Email: ong@cs.wisc.edu
[3] Department of Genetics, University of Wisconsin
Madison, WI 53706
Email: jalewis4@wisc.edu

Abstract. Transcriptional regulation play an important role in every cellular decision. Gaining an understanding of the dynamics that govern how a cell will respond to diverse environmental cues is difficult using intuition alone. We introduce logic-based regulation models based on state-of-the-art work on statistical relational learning, to show that network hypotheses can be generated from existing gene expression data for use by experimental biologists.

1 Introduction

Transcriptional regulation refers to how proteins control gene expression in the cell. Many major cellular decisions involve changes in transcriptional regulation. Thus, gaining insight into transcriptional regulation is important not just for understanding the fundamental biological processes, but also will have deep practical consequences in fields such as the medical sciences. With the advent of high-throughput technologies and advanced measurement techniques molecular biologists and biochemists are rapidly identifying components of transcriptional networks and determining their biochemical activities. Unfortunately, understanding these complex multicomponent networks that govern how a cell will respond to diverse environmental cues is difficult using intuition alone.

In this work, we aim at building probabilistic logical models thatwould uncover the structure and dynamics of such networks and how they regulate their targets.

Despite the challenge of inferring genetic regulatory networks from gene expression data, various computational models have been developed for regulatory network analysis. Examples include approaches based on logical gates [1, 2], and probabilistic approaches, often based on Bayesian networks [3]. On one hand, logic gates provide a natural, intuitive way to describe interactions between

proteins and genes. On the other hand, probabilistic approaches can handle incomplete and imprecise data in a very robust way.

Our main contribution is in introducing a model that combines the two approaches. Our approach is based on the probabilistic logic programming language ProbLog [4, 5]. In this language, we can express true logical statements (expressed as *true rules*) about a world where there is uncertainty over data, expressed as *probabilistic facts*. In the setting of gene expression, this corresponds to establishing:

(1) a set of *true rules* describing the possible interactions existing in a cell;
(2) a set of *uncertain facts* describing which possible rules are applicable to a certain gene or set of genes.

Given time-series gene expression data, we want to choose the probability parameters that best describe the data. Our approach is to reduce this problem to an optimization problem, and use a gradient ascent algorithm to estimate a local solution [6] in the style of logistic regression. We further contribute an efficient implementation to this algorithm that computes both probabilities and gradients through binary decision diagrams (BDD). We validate our approach by using it to study expression data on an important gene-expression pathway, the Hog1 pathway [7].

Related Work Logic-based modeling is seen as an approach lying midway between the complexity and precision of differential equations on one hand and data-driven regression approaches on the other[8].

Despite the difficulty of deciphering genetic regulatory networks from microarray data, numerous approaches to the task have been quite successful. Friedman *et al.* [3] were the first to address the task of determining properties of the transcriptional program of *S. cerevisiae* (yeast) by using Bayesian networks (BNs) to analyze gene expression data. Pe'er *et al.* [9] followed up that work by using BNs to learn master regulator sets. Other approaches include Boolean networks (Akutsu *et al.* [10], Ideker *et al.* [11]) and other graphical approaches (Tanay and Shamir [12], Chrisman *et al.* [13]).

The methods above can represent the dependence between interacting genes, but they cannot capture causal relationships. In our previous work [14], we proposed that the analysis of time series gene expression microarray data using Dynamic Bayesian networks (DBNs) could allow us to learn potential causal relationships.

2 Methods

Recently, there has been interest in combining logical and probabilistic representations within the framework of Statistical Relational Learning [15]. This framework allows the compact representation of complex networks, and has been implemented over a large variety of languages and systems. Arguably, one of the most popular SRL languages is the programming language ProbLog [4, 5]. This

language was initially motivated by the problem of representing a graph where there is uncertainty over whether edges exist or not. As a straightforward example consider the directed graph in Figure 1.

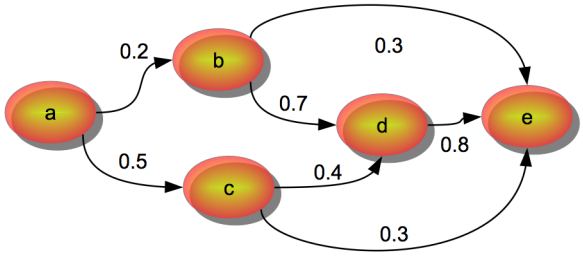


Fig. 1. A simple directed graph, where each edge has a probability of being true.

Notice that each edge has a probability of being true. As an example, starting from `a` we can reach `b` with probability 0.2 and `c` with probability 0.5. We assume that all the different probabilities are *independent*.

Given the example in Fig 1, ProbLog allows one to answer several queries, such as *what is the most likely path between two nodes*, and *what is the total probability that there is a path between two nodes*. The algorithm takes advantage of independence between probabilistic facts.

Note that computing the probability is not simply the sum if different paths have a common edge. As an example, consider $Pr(\texttt{ae})$. The path `abde` shares the edge `de` with `acde`, and the edge `ab` with `abe`. Summing these three paths would count two edges twice.

Kimmig and de Raedt proposed an effective solution to this problem. The idea is that probability can be computed as a sum if the paths do not share edges. This can be obtained by selecting an edge (or fact), and splitting into the case where the edge is true and the case where the edge is false. The process can be repeated recursively until we run out of facts to split. Kimmig and de Raedt's key observation is that this idea is indeed the same one that is used to construct binary decision diagrams (BDDs): the total probability can be obtained by generating a BDD from the proofs.

Binary decision diagrams provide a very efficient implementation for probability computation over small and medium graphs. Unfortunately, they do not scale to larger graphs with thousands of nodes. In this case, ProbLog implementations rely on approximated solutions, either Monte Carlo methods or often by approximating the total probability by the probability of the best k queries [5].

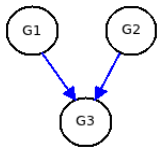
3 Experimental Methodology

We obtained time-series gene expression data from Lee et al. [16] for our experiments. The experiments followed the response of actively growing Saccharomyces cerevisiae to an osmotic shock of 0.7 M NaCl. The dose of salt was selected by the experimentalists to provide a robust physiological response but allow high viability and eventual resumption of cell growth. The samples were collected

before and after NaCl treatment at 30, 60, 90, 120, and 240 min (measuring the peak transcript changes that occurs at or after 30 min) [17]. We focused our attention on the 270 genes of the Hog1 Msn2/4 pathway from Capaldi [7] for which we have expression data and utilized the temporal data to better estimate the relationships from the data.

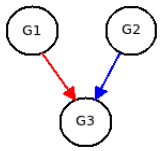
Our experiments aim for a more detailed picture of the learned network by using the temporal nature of the data. The output generated is a weighted, directed gene network, but nodes are connected as a gated network:

– **AND:** two promoter genes need to be active in order to activate a gene, as shown in the graph. We also show the ProbLog code for the temporal model:



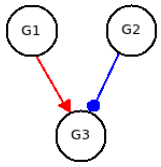

```
active(G3,T1,Z) :-
        next_step(T0,T1),
        and(G1,G2,G3),
        active(E,T0,G1),
        active(E,T0,G2).
```

– **OR:** either promoter gene needs to be active in order to activate a gene, as shown in the graph. We also show the corresponding ProbLog code for the temporal model.




```
active(G3,T1,Z) :-
        next_step(T0,T1),
        or(G1,G2,G3),
        active(E,T0,G1).
active(G3,T1,Z) :-
        next_step(T0,T1),
        or(G1,G2,G3),
        active(E,T0,G2).
```

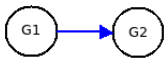
– **XOR:** one promoter gene needs to be active and one repressor gene needs to be inactive in order to activate a gene, as shown in the graph.




```
active(G3,T1,Z) :-
        next_step(T0,T1),
        xor(G1,G2,G3),
        active(E,T0,G1),
        not_active(E,T0,G2).
```

This is the only case where we allow the possibility of negative regulation.

– **SINGLE:** a unique promoter gene regulates the target gene.




```
active(G2,T1,Z) :-
        next_step(T0,T1),
        single(G1,G2),
        active(E,T0,G1).
```

We use two different forms of temporal data: expression level (E), and variation (Δ). We experimented with three different approaches:

(1) Level influences variation (LV).
(2) Variation influences variation (VV).
(3) Level influences level (LL).

One important advantage of the approach is that it allows us to implement *soft constraints* on the probability distribution. These constraints are implemented by saying that satisfying some rule must have probability 1 or 0. In our experiments, we implement constraints saying that a *gene must be explained by a single rule*. Two example constraints for **OR** are of the form: The next constraint says that there must be a single set of parents for a gene defined with the LV $\vee$ rule:

$$\begin{array}{c} E_t(G_1) \vee E_t(G_2) \Rightarrow \Delta_{t+1}(G) \\ \wedge \\ E_t(G_3) \vee E_t(G_4) \Rightarrow \Delta_{t+1}(G) \\ \rightarrow \\ G_1 = G_3 \wedge G_2 = G_4 \end{array}$$

The second constraint ensures that we cannot use two rules of different types at the same time:

$$\begin{array}{l} \neg(\ E_t(G_1) \vee E_t(G_2) \Rightarrow \Delta_{t+1}(G) \\ \qquad \wedge \\ \quad E_t(G_3) \oplus E_t(G_4) \Rightarrow \Delta_{t+1}(G) \\ \ ) \end{array}$$

In practice, we must be careful not to flood the system with soft constraints. In our experiment we implemented one joint soft constraint per gene.

4 Conclusion

Learning regulatory networks from gene expression is a hard problem. Data is noisy and relationships between genes highly complex. We present a statistical relational approach to modeling pathways. Our approach allows us to design a coarser and a more fine grained model, based on probabilistic gates.

We plan to continue improving the model quality and experiment with new data. Specifically, we would like to experiment with implementing a regression based approach, as it fits our framework naturally. Last, but not least, we would like to investigate how to reduce the number of parameters in the model by exploiting strong correlations between gene expression.

Acknowledgments

This work is financed by the ERDF European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) FCOMP-01-0124-FEDER-010074 and by National Funds through the FCT Fundacão para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project HORUS (PTDC/EIA-EIA/100897/2008) and by the US 760 Department of Energy (DOE) Great Lakes Bioenergy Research Center (DOE BER 761 Office of Science DE-FC02-07ER64494).

References

1. Glass, L., Kauffman, S.: A logical analysis of continuous, non-linear biochemical control networks. Journal of Theoretical Biology **39** (1973) 103–129
2. Thomas, R.: Boolean formalization of genetic control circuits. Journal of Theoretical Biology **42** (1973) 563–585
3. Friedman, N., Linial, M., Nachman, I., Pe'er, D.: Using Bayesian networks to analyze expression data. Journal of Computational Biology **7**(3/4) (2000) 601–620
4. Raedt, L.D., Kimmig, A., Toivonen, H.: Problog: A probabilistic prolog and its application in link discovery. In Veloso, M.M., ed.: IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007. (2007) 2462–2467
5. Kimmig, A., Santos Costa, V., Rocha, R., Demoen, B., Raedt, L.D.: On the Implementation of the Probabilistic Logic Programming Language ProbLog. Theory and Practice of Logic Programming Systems **11** (2011) 235–262
6. Gutmann, B., Kimmig, A., Kersting, K., Raedt, L.D.: Parameter learning in probabilistic databases: A least squares approach. In: ECML/PKDD–08. Volume LNCS 5211., Antwerp, Belgium, Springer (September 15–19 2008) 473–488
7. Capaldi, A., Kaplan, T., Liu, Y., Habib, N., Regev, A., Friedman, N., O'Shea, E.: Structure and function of a transcriptional network activated by the mapk hog1. Nature Genetics **40** (2008) 1300–1306
8. Morris, M., Saez-Rodriguez, J., Sorger, P., Lauffenburger, D.: Logic-based models for the analysis of cell signaling networks. Biochemistry **49** (2010) 3216–3224
9. Pe'er, D., Regev, A., Tanay, A.: Minreg: Inferring an active regulator set. In: Proceedings of the 10th International Conference on Intelligent Systems for Molecular Biology, Oxford University Press (2002) S258–S267
10. Akutsu, T., Kuhara, S., Maruyama, O., Miyano, S.: Identification of gene regulatory networks by strategic gene disruptions and gene overexpressions. In: Proc. the 9th Annual ACM-SIAM Symposium on Discrete Algorithms. (1998) 695–702
11. Ideker, T., Thorsson, V., Karp, R.: Discovery of regulatory interactions through perturbation: Inference and experimental design. In: Pacific Symposium on Biocomputing. (2000) 302–313
12. Tanay, A., Shamir, R.: Computational expansion of genetic networks. In: Bioinformatics. Volume 17. (2001)
13. Chrisman, L., Langley, P., Bay, S., Pohorille, A.: Incorporating biological knowledge into evaluation of causal regulatory hypotheses. In: Pacific Symposium on Biocomputing (PSB). (January 2003)
14. Ong, I., Glasner, J., Page, D.: Modelling regulatory pathways in *Escherichia coli* from time series expression profiles. Bioinformatics **18** (2002) S241–S248
15. Taskar, B., Getoor, L.: Introduction to Statistical Relational Learning. MIT Press (2007)
16. Lee, V., Topper, S., Hubler, S., Hose, J., Wenger, C., Coon, J., Gasch, A.: A dynamic model of proteome changes reveals new roles for transcript alteration in yeast. Molecular Systems Biology **7**(514) (2011)
17. Berry, D.B., Gasch, A.P.: Stress-activated genomic expression changes serve a preparative role for impending stress in yeast. Molecular Biology of the Cell **19**(11) (2008) 4580–4587

Creative Problem Solving by Concept Generation Using Relation Structure

Katsutoshi Kanamori and Hayato Ohwada

Tokyo University of Science, Chiba, Japan
{katsu, ohwada}@rs.tus.ac.jp

Abstract. The purpose of our work is to achieve creative knowledge processing. In this paper, we focus on the formulation of concept generation and its use in problem solving. We propose a method for solving a problem by generating new concepts that have never appeared in existing knowledge. We propose Creative Problem Solving, which can derive a goal state by using a creative leap invoked by concept generation.

1 Introduction

Our work seeks to achieve creative knowledge processing. Some studies have been conducted on computational creativity and theory invention in the AI community[1]. The intention of this paper is to formulate concept generation based on logic, and to investigate problem solving with concept generation. Only a few attempts have been made at such a study. There are studies Predicate Invention in ILP,[2, 3] but our focus was not only on induction, but rather on developing a general method of concept generation. This concept generation constitutes a new approach to problem solving, addressing problems that induction cannot solve. An early study described an AM[4] concept generation system cannot be considered a general concept generation.

This paper proposes a formulation of problem solving by generating new concepts that have never appeared in existing knowledge, and we confirm that such knowledge processing is implementable. We call this kind of processing Creative Problem Solving, and consider it is a part of creative knowledge processing.

2 Preliminaries

2.1 Knowledge and Relation Structures

We are concerned with first-order logic as representing Knowledge. Any logical formulae can be transformed to the formulae include no functions and no individual constants. We may regard concept generation as predicate generation.

In this paper, knowledge is defined as a set of logical formulae with no functions and no individual constants. A relation structure is defined as a logical formula that has at least one predicate variable. A relation structure also has no functions and no individual constants.

For example, Let $S_1 = \{\forall x, y : \neg P_1(x) \vee P_2(x, y)\}$, $S_2 = \{\forall x, y : \neg X_1(x) \vee P_2(x, y)\}$. When P_1, P_2 are predicate constants and X_1 is a predicate variable, S_1 is knowledge, and S_2 is a relation structure.

2.2 Simple Substitution

We define the substitution replace predicate variables with predicate constants.

Let P_{vn} be a universal set of free predicate variables of arity n, and let P_{cn} be a universal set of invariable predicate of arity n. If Θ satisfies $\Theta \subset P_{v1} \times P_{c1} \cup \cdots \cup P_{vi} \times P_{ci}$, and each variable and constant that occurs in Θ is distinct, then we say Θ is a *simple substitution*D The element of Θ is written in a manner similar to the style of the general substitution, v/c. Here, v is the variable and c is the constant.

For example, relation structure $S_1 = \{\forall x : X_1(x, y) \wedge X_2(x)\}$ and simple substitution $\Theta_1 = \{X_1/A, X_2/B\}$ are given, then $S_1\Theta_1 = \{\forall x : A(x, y) \wedge B(x)\}$.

3 Concept Generation

3.1 Predicate Generation

When knowledge Σ, relation structure RS, and a predicate variable X (which occurs in RS) are given, we define new knowledge S_{NEW} that holds a new predicate. If a simple substitution Θ has all predicate variables that occur in RS except X, then S_{NEW} is defined as follows : $S_{NEW} = RS(\Theta \cup \{X/NEW\})$.

The new predicate NEW is obtained by generating new knowledge S_{NEW}. Here, the generated S_{NEW} is determined uniquely by Σ, RS, X, Θ, and new symbol NEW. Therefore we can regard predicate generation in terms of sets of these five tuple $(\Sigma, RS, X, \Theta, NEW)$.

3.2 Characteristics of New Knowledge

Novelty Novelty is a property representing whether new knowledge is obtained as a logical conclusion based on existing knowledge or not. If S_{NEW} satisfies the condition that : for all s such that $S_{NEW} \models s$ and s includes NEW and $\Sigma \not\models s$, then the predicate generation is said to possess novelty.

Consistency Consistency is the property by which new knowledge and existing knowledge are not in contradiction. With consistent predicate generation, we can take on $\Sigma \cup S_{NEW}$ as new knowledge instead of S_{NEW}.

Soundness To be sound means that $\Sigma \cup S_{NEW}$ is consistent with all logical formulae which are consistent with Σ and has no new predicates. If this condition is true, then we say that the predicate generation is sound or S_{NEW} is sound.

3.3 Example of Predicate Generation

Let Σ be knowledge and RS be a relation structure as follows:

$$\Sigma = \{\forall x : \neg bird(x) \vee ab(x) \vee fly(x)\},$$

$$RS = \left\{ \begin{array}{l} \forall x : \neg X_0(x) \vee X_1(x) \vee X_2(x) \\ \forall x : \neg X_3(x) \vee X_0(x) \\ \forall x : \neg X_3(x) \vee \neg X_2(x) \end{array} \right\}$$

knowledge Σ describes that birds fly if not ab. See predicate variable X_3, and consider two simple substitutions for predicate generation.

$$\Theta_1 = \{X_0/bird, X_1/ab, X_2/fly\}$$

$$\Theta_2 = \{X_0/fly, X_1/bird, X_2/ab\}$$

Then, predicate generations $(\Sigma, RS, X_3, \Theta_1, NEW_1)$, $(\Sigma, RS, X_3, \Theta_2, NEW_2)$ generate new knowledge S_{N_1}, S_{N_2} as follows:

$$S_{N_1} = \left\{ \begin{array}{l} \forall x : \neg bird(x) \vee ab(x) \vee fly(x) \\ \forall x : \neg NEW_1(x) \vee bird(x) \\ \forall x : \neg NeW_1(x) \vee \neg fly(x) \end{array} \right\}$$

$$S_{N_2} = \left\{ \begin{array}{l} \forall x : \neg fly(x) \vee bird(x) \vee ab(x) \\ \forall x : \neg NEW_2(x) \vee fly(x) \\ \forall x : \neg NEW_2(x) \vee \neg ab(x) \end{array} \right\}$$

NEW_1 is a new concept means such as "flightless bird", NEW_2 means "not ab and fly". S_{N_1} has novelty and are consistent and sound. S_{N_2} has novelty and is consistent but not sound. In fact, if $l = \neg(\forall x : \neg fly(x) \vee bird(x) \vee ab(x))$, then $\Sigma \wedge l$ is consistent, but $(\Sigma \cup S_{N_2}) \wedge l$ is not.

4 Expand Dimension

Even if the predicate generation is inconsistent, it can be useful in expanding the knowledge dimension. For example, consider adding a concept describing imaginary numbers to real-number knowledge. The rule "the square of any number greater than or equal to 0" is inconsistent with the new knowledge, but serves to expand the dimension of real numbers to complex numbers, making it possible to regard new knowledge as consistent knowledge. This is a method for expanding the knowledge dimension naturally by predicate generation

Now, if $C_\Sigma \subseteq \Sigma$ and $(\Sigma - C_\Sigma) \cup S_{NEW}$ is consistent and $C_\Sigma \cup S_{NEW}$ is inconsistent, consider C'_Σ as follows.

$$C'_\Sigma = \left\{ \begin{array}{l} \neg inD(x_{11}) \vee \cdots \vee \neg inD(x_{1l_1}) \vee s'_1 \\ \vdots \\ \neg inD(x_{n1}) \vee \cdots \vee \neg inD(x_{nl_n}) \vee s'_n \end{array} \right\}$$

inD is a new predicate prepared here. $s_1, \cdots, s_n$ are elements of C_Σ, and $x_{j1}, \cdots, x_{jl_j}$ are all bind variables occuring in s_j. $s'_1, \cdots, s'_n$ are formulae obtained by transforming $s_1, \cdots, s_n$ to prenex normal form and cutting out the head part. Each formula for C'_Σ has a head part $q_{j1}x_{j1} \cdots q_{jl_1}x_{jl_1}$ that is the head part of the corresponding s_j, although these are omitted here. q_{jk} is $\exists$ or $\forall$, equal to a qualifier of s_j. We can imagine that $inD(x)$ means x is in a dimension of former knowledge. $(\Sigma - C_\Sigma) \cup C'_\Sigma \cup S_{NEW}$ is then consistent.

Theorem 1. *For predicate generation $(\Sigma, RS, X, \Theta, NEW)$, if Σ and RS are consistent then $\Sigma' = (\Sigma - C_\Sigma) \cup C'_\Sigma \cup S_{NEW}$ is consistent.*

5 Creative Leap

For predicate generation, it is important to consider whether the new knowledge represents a leap or not. The leap enable us to achieve our creative goal.

In defining a creative leap, it is important to consider whether new predicates occur in generated consequence from the conjunction of new knowledge and existing knowledge. Formulae that have new predicates, cannot belong to unknown facts, so they are not appropriate as leap conclusions, because a new predicate is made by the system on its own.

Therefore, we define a Creative Leap as the case where there exist logical formulae not having NEW, such that $\Sigma \cup S_{NEW} \models s$ and $\Sigma \not\models s$. Then, we say the predicate generation leaps.

Theorem 2. *If predicate generation is not sound, then it leaps.*

6 Creative Problem Solving

When a goal state, which is not derived from existing knowledge and it's negation is not derived either, is given, predicate generation with a creative leap can lead to the goal state as a consequence. Even if existing knowledge derives the negation of the goal state, expanding of dimensions enable to lead to the goal state. We call such problem solving Creative Problem Solving.

For example, human kind had wanted to fly like a bird. And known that birds can fly and the wing enables them, so create new concept like a wing.

Example 1.

$$\Sigma_f = \left\{ \begin{array}{l} \forall x \exists y : Bird(x) \rightarrow Has(x, y) \wedge Wing(y) \\ \forall x : Human(x) \rightarrow \neg(\exists y : Has(x, y) \wedge Wing(y)) \\ \forall x, y : Has(x, y) \wedge WingBehavior(y) \rightarrow Fly(x) \\ \forall x : Wing(x) \rightarrow WingBehavior(x) \end{array} \right\}$$

is given, and goal $G_f = \forall x : Human(x) \rightarrow Fly(x)$. Then Σ_f cannot lead to G_f as a consequence. Then we consider RS as follows:

$$RS_f = \left\{ \begin{array}{l} \forall x \exists y : X_1(x) \rightarrow X_2(x,y) \wedge X_3(y) \\ \forall x,y : X_2(x,y) \wedge X_4(y) \rightarrow X_5(x) \\ \forall x : X_3(x) \rightarrow X_4(x) \end{array} \right\}$$

The simple substitution Θ_f as follows:

$$\Theta_f = \{X_1/Human, X_2/Has, X_4/WingBehavior, X_5/Fly\}$$

Predicate generation $(\Sigma_f, RS_f, X_3, \Theta_f, NEW_f)$ then generates S_{NEW_f} as follows :

$$S_{NEW_f} = \left\{ \begin{array}{l} \forall x \exists y : Human(x) \rightarrow Has(x,y) \wedge NEW_f(y) \\ \forall x,y : Has(x,y) \wedge WingBehavior(y) \rightarrow Fly(x) \\ \forall x : NEW_f(x) \rightarrow WingBehavior(x) \end{array} \right\}$$

S_{NEW_f} possesses novelty and is consistent but not sound. S_{NEW_f} derives G_f. G_f was not obtained by Σ_f, therefore, a new predicate is generated to derive the goal state by using a creative leap. NEW_f was not generated randomly, and may fulfill $WingBehavior$ like a bird's wing.

6.1 Practical Method of Predicate Generation

When a goal state is given, what prepared for generation is Σ_f only. The important topic is how to prepare a relation structure and which simple substitutions and predicate variables to choose. The problem of what predicate should be generated and what relation structure should be prepared is differs for each purpose and each case. We show a general method of extracting relation structures from knowledge, and present one of the methods for preparing RS, Θ, and X.

The relation structure can be gained by generalizing knowledge. Concretely, replacing all or some predicate constants occurring in knowledge (or a subset) with free predicate variables.

Next, the point is which RS, Θ, and X are chosen. The destination is to derive G from $\Sigma \cup S_{NEW}$. We propose a method of extracting relation structure from an explanation structure inherent in existing knowledge.

Now, for a knowledge σ, let $RS(\sigma)$ be a relation structure obtained by replacing all of the predicate constants in σ with each independent predicate variables.

If there exists Σ^* such that $\Sigma^* \subset \Sigma : \exists\Theta : RS(\Sigma^*)\Theta \models RS(G)\Theta$, then we can consider predicate generation using $RS(\Sigma^*)$ as a relation structure. This $RS(\Sigma^*)$ is an explanation structure that derives a result with the same structure as G. New knowledge possossing the same explanation structure may be obtained by using another simple substitution to derive G. It can be thought this method is based on analogy as common structure.

Example 2. Only knowledge Σ_f and goal state G_f are given. Here, $RS(G_f) = \{\forall x : X_1(x) \rightarrow X_2(x)\}$, and Σ_f^* including $RS(G_f)$, can be taken :

$$\Sigma_f^* = \left\{ \begin{array}{l} \forall x \exists y : Bird(x) \rightarrow Has(x,y) \wedge Wing(y) \\ \forall x,y : Has(x,y) \wedge WingBehavior(y) \rightarrow Fly(x) \\ \forall x : Wing(x) \rightarrow WingBehavior(x) \end{array} \right\}$$

so we can obtain $RS(\Sigma_f^*)$ in concert with $RS(G_f)$ as : $RS(\Sigma_f^*) = RS_f$.

Let $\Theta = \{X_1/Bird, X_2/Fly, X_3/Has, X_4/Wing, X_5/WingBehavior\}$, then $RS(\Sigma_f^*)\Theta \models RS(G_f)\Theta$. Define simple substitution Θ' as follows : $\Theta' = \Theta_f$. Then, predicate generation $(\Sigma_f, RS(\Sigma_f^*), \Theta', X_4, NEW_f)$ produces $S'_{NEW} : S'_{NEW} = S_{NEW_f}$. Leap is realized and goal state G_f is derived from S'_{NEW}.

7 Conclusion

This paper proposed a formulation of concept generation, and an approach to creative problem solving. A creative leap shows the new predicate is not a mere paraphrase but achieves a new result.

We may note here that creative problem solving is similar to abduction in the sense that it is a method of deriving a goal by generating new knowledge. If the goal state is observed facts, we can regard creative problem solving as abduction accompanied by concept generation. However, creative problem solving is a more general method.

Our example of creative problem solving was the simulation of the invention of something like an airplane. Though not every invention can be described using this frame, it is very interesting that creative problem solving can simulate a part of human's creative knowledge processing and invention.

However, many problems need to be solved to develop a practical system with creative problem solving. The problem of calculating of new-knowledge consistency, determining which structures and predicate variables are important, and selecting simple substitutions, among other factors, are important for practical systems. Some of these problems (perhaps most of them) are depend on specific systems.

We showed logically that concept generation may achieve creative knowledge processing. We expect to achieve a fully creative system in future work.

References

1. Simon Colton and Ramon López de Mántaras and Oliviero Stock: Computational Creativity: Coming of Age. AI Magazine vol.30 num.3. (2009) 11–14
2. S.H. Muggleton., and Buntime, R.: Machine Invention of first-order Predicates by Inverting Resolution. the 5th Intl. Workshop on Machine Learning, ANN Arbor, MI. (1988) 339–352
3. S.H. Muggleton.: Predicate invention and utilisation. Journal of Experimental and Theoretical Artificial Intelligence, 6(1). (1994) 127–130
4. Davis, R., Lenat, D .B.: Knowledge-Based System in Artificial Intelligence. McGraw-Hill, NewYork (1982)

On Computing Minimal Generators in Multi-Relational Data Mining with respect to θ-Subsumption

Noriaki Nishio, Atsuko Mutoh, and Nobuhiro Inuzuka

Nagoya Institute of Technology,
Gokiso-cho, Showa, Nagoya 466-8555, Japan
nishio@nous.nitech.ac.jp, mutoh@nitech.ac.jp, inuzuka@nitech.ac.jp

Abstract. We study the minimal generators (mingens) in multi-relational data mining. The mingens in formal concept analysis are the minimal subsets of attributes that induce the formal concepts. An intent for a formal concept is called a closed pattern. In contrast to the wide attention to closed patterns, the mingens have been paid little attention in Multi-Relational Data Mining (MRDM) field. We introduce an idea of non redundant mingens in MRDM. The notion of mingens in MRDM is led by θ-subsumption relation among patterns, and is useful to grasp the structure and information in the concepts.

1 Introduction

Formal Concept Analysis (FCA) [1] is an important tool for data analysis and knowledge discovery [2]. A formal concept is determined by its extent and its intent. The intent of a formal concept is the closure of the attributes, itemsets, or patterns that form a maximum characterization of the formal concept. Mining the closed patterns [3, 4] has attracted a lot of attentions because it reduces the number of patterns by selecting only representative patterns of their equivalent patterns in the sense that they produce the same extent.

While a closure is the maximal pattern presenting a concept, a minimal generator (mingen) [5] is a minimal pattern. The mingens play an important role in many contexts, e.g., database design (as key sets), graph theory (as minimal transversals), and data mining (as minimal premises of association rules). Dong et al. study the mingens and define Succinct System of Mingens (SSMG) [6] which removes redundant mingens. In this paper, we state that SSMGs of a formal context for relational patterns have further redundancy and propose a novel concept of non redundant mingens based on θ-subsumption of Multi-Relational Data Mining (MRDM) [7].

Sections 2 and 3 introduce FCA and MRDM. Section 4 describes about mingens. Section 5 provides a definition of minimal generators consisted of relational patterns. Then section 6 reports experimental results on compactness.

	a	b	c	d	e	g	h	i
t_1	×	×	×	×	×	×	×	×
t_2	×		×	×		×		
t_3		×	×	×		×	×	×
t_4	×	×		×			×	×
t_5		×	×		×	×	×	×

Fig. 1. A context that involves relations between objects and attributes.

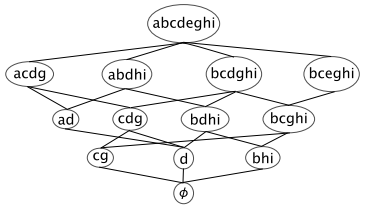


Fig. 2. A concept lattice for the context of Fig. 1.

2 Formal Concept Analysis

We review the basis of Formal Concept Analysis. Start with an arbitrary relation, $I \subseteq G \times M$, between G, a set of objects, and M, a set of attributes, and define

$$A \mapsto A^I = \{m \in M \mid (g, m) \in I \text{ for all } g \in A\} \text{ for } A \subseteq G,$$
$$B \mapsto B^I = \{g \in G \mid (g, m) \in I \text{ for all } m \in B\} \text{ for } B \subseteq M.$$

A triple $\mathbb{K} = (G, M, I)$ is called a *formal context.*

Definition 1 (formal concept). *A pair* (X, Y) *is called a* formal concept *of a* formal context $\mathbb{K} = (G, M, I)$*, if it satisfies*

$$X \subseteq G, Y \subseteq M, X^I = Y, X = Y^I. \qquad \square$$

When we define an order by $(X_1, Y_1) \leq (X_2, Y_2) \Longleftrightarrow X_1 \subseteq X_2$ ($\Leftrightarrow Y_2 \subseteq Y_1$), among formal concepts of a formal context $\mathbb{K}$, it forms a complete lattice. We call it the *concept lattice* of $\mathbb{K}$.

Example 1. A Fig. 1 shows a formal context where each object has a set of attributes. A pair $(t_1t_2t_3, cdg)$ (set brackets are omitted) is a formal concept, where $t_1t_2t_3^I = cdg$ and $cdg^I = t_1t_2t_3$. Fig. 2 shows the concept lattice. Each concept is labeled by its intent. $\square$

3 Multi-Relational Data Mining

While propositional data mining algorithms look for patterns in a single data table, MRDM algorithms look for relational patterns, represented by logical formulae, that involve multiple tables (relations).

Example 2. A Database $R_{\mathbf{fam}}$ in Fig. 3 includes four relations on families, where $\mathsf{grandfather}(x)$ meaning x is someone's grandfather, $\mathsf{parent}(x, y)$ meaning x is a parent of y, $\mathsf{male}(x)$ for male x, and $\mathsf{female}(x)$ for female x. Then a pattern, such as $\mathsf{grandfather}(X) \leftarrow \mathsf{parent}(X, Y), \mathsf{parent}(Y, Z), \mathsf{female}(Z)$, can be found. $\square$

grandfather
person01
person07
person12
person19
person20

parent	
person01	person02
person02	person03
person02	person04
person03	person05
...	...

male
person01
person05
person07
person10
...

female
person02
person03
person04
person06
...

Fig. 3. The family DB $R_{\mathbf{fam}}$, including grandfather, parent, male and female.

4 Succinct System of Mingens

The mingens are defined bellow.

Definition 2 (mingens). *A set $P \subseteq M$ is called a* mingen *for a formal concept (X, Y) of a formal context $\mathbb{K} = (G, M, I)$ if $P^I = X$ but for every proper subset $P' \subset P$, $P'^I \neq X$.* □

Example 3. In Fig. 1, bc, bg, ch, ci, gh, and gi are mingens for a formal concept $(t_1t_3t_5, bcghi)$. □

In the above example, the closed itemset $bcghi$ has six mingens, where b, h and i always appear together in each object and thus can be exchanged each other, and similarly for c and g. An SSMG is a representative of each equivalence class, which is defined bellow. A criterion which selects a representative is left to users, because dependence between items is not defined.

Definition 3 (C-equivalence). *X, $Y \subseteq M$ are* C-equivalent *for a formal concept C of a formal context $\mathbb{K} = (G, M, I)$, denoted $X \approx_C Y$, if they satisfy either following condition.*

1. *There is a concept $C' \geq C$ such that both X and Y are mingens of C'.*
2. *There are subsets $Z, Z', M \subseteq M$ such that $X = W \cup Z$, $Y = W \cup Z'$, and $Z \approx_C Z'$.* □

Definition 4 (SSMG). *Given an order $\sqsubseteq$ on M, a* succinct system of mingens *(SSMG) by the order $\sqsubseteq$ for a formal concept C of a formal context $\mathbb{K}$ consists of elements satisfied either following condition.*

1. *If C is a maximal formal concept in the sense of the concept lattice except $(G, \emptyset)$, an SSMG for C holds the following conditions.*
   - *It is a mingen for C.*
   - *It is minimal in the sense of $\sqsubseteq$ among all mingens in a C-equivalence class for C.*
2. *Otherwise, a mingen in a C-equivalence class is an SSMG for C if it does not include any mingen which is not an SSMG for $C' \geq C$.* □

The definition of SSMGs in [6] uses the alphabetic lexicographic order for $\sqsubseteq$ above. Since the alphabetic order is linear, there is a unique SSMG for an equivalence class. Our extended definition allows a partial order and then there are more than one SSMGs.

Table 1. Attributes by expressed formulae.

$a = \mathsf{gf}(A) \leftarrow \mathsf{m}(A).$
$b = \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B), \mathsf{f}(B).$
$c = \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B), \mathsf{p}(B,C), \mathsf{f}(C).$
$d = \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B), \mathsf{p}(B,C), \mathsf{p}(C,D), \mathsf{f}(D).$
$e = \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B), \mathsf{p}(B,C), \mathsf{p}(C,D), \mathsf{m}(D).$
$f = \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B), \mathsf{f}(B), \mathsf{p}(B,C), \mathsf{f}(C)$
$g = \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B), \mathsf{f}(B), \mathsf{p}(B,C), \mathsf{f}(C), \mathsf{p}(C,D), \mathsf{m}(D).$
$h = \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B), \mathsf{p}(B,C), \mathsf{f}(C), \mathsf{p}(C,D), \mathsf{p}(D,E), \mathsf{m}(E).$
$i = \mathsf{gf}(A) \leftarrow \mathsf{p}(A,B), \mathsf{p}(B,C), \mathsf{p}(C,D), \mathsf{f}(D), \mathsf{p}(B,E), \mathsf{p}(E,F), \mathsf{m}(F).$

	a	b	c	d	e	f	g	h	i
gf(01)	×	×	×	×	×	×	×	×	×
gf(07)	×	×	×	×	×	×	×	×	×
gf(12)	×	×	×		×	×	×	×	
gf(19)	×		×	×	×			×	×
gf(20)	×	×	×			×			

Fig. 4. $\mathbb{K}'_{\mathbf{fam}} = (G, M, I)$ w.r.t $R_{\mathbf{fam}}$

Table 2. Formal concepts of $\mathbb{K}'_{\mathbf{fam}}$

(G, ac)
$(\mathsf{gf}(01)\mathsf{gf}(07)\mathsf{gf}(12)\mathsf{gf}(19), aceh)$
$(\mathsf{gf}(01)\mathsf{gf}(07)\mathsf{gf}(12)\mathsf{gf}(20), abcf)$
$(\mathsf{gf}(01)\mathsf{gf}(07)\mathsf{gf}(19), acdehi)$
$(\mathsf{gf}(01)\mathsf{gf}(07)\mathsf{gf}(12), abcefgh)$
$(\mathsf{gf}(01)\mathsf{gf}(07), M)$

5 Mingens of MRDM

Though SSMGs remove redundant patterns, a simple application of SSMGs to relational patterns does not remove all of redundancy. Mapix [8, 9], a miner in MRDM, enumerates patterns consisted of *property items* [8] which are restricted sets of literals. Though a search space of MRDM has no limit as long as adding literals, that of Mapix restricts into a meaningful form by modes of predicates.

We construct a formal context $\mathbb{K}' = (G, M, I)$ of property items produced by Mapix, which we call a *formal relational context*, where G is a set of instances of a target (key) relation (e.g., grandfather relation in Fig. 3), M is a set of property items (e.g., in Table 1), and I is relation among G and M, which indicates whether an instance satisfies a property item (e.g., $\mathbb{K}'_{\mathbf{fam}}$ in Fig. 4). The notion of the formal relational context was discussed in [10]. Then we can also compute formal concepts and SSMGs of the formal relational context $\mathbb{K}'$.

Logical Mingens We reduce further redundancy of SSMGs in relational patterns by θ-subsumption relation, i.e. C θ-subsumes D, denoted by $C \succeq D$, if $C\theta \subseteq D$, for a substitution θ, where C and D are clauses.

Definition 5 (LMG). *A mingen for a formal concept C of a formal relational context $\mathbb{K}'$ is called a* logical mingen *(LMG) for C of $\mathbb{K}'$, if it satisfies the following conditions.*

- *It is an SSMG by θ-subsumption order ($\preceq$).*
- *It is a minimal in the sense of $\preceq$ in among all SSMG in C-equivalence class.*

□

Table 3. LMG_MINER : the algorithm for enumerating LMG

LMG_MINER($\mathbb{K}'$, $\sup_{\min}$):

input $\mathbb{K}'$: A formal relational context;
$\sup_{\min}$: A support threshold;

output LMG : logical mingens;

1. **let** $LMG := \emptyset$;
2. **let** $I :=$ {all attributes associated with property items};
3. **let** $LC :=$ {items occurring in all transactions};
4. **call** $DFS(H := \emptyset, T := I - LC, LC)$;
5. **return** LMG;

$DFS(H, T, LC)$:

1. **if** $\sup(H) < \sup_{\min}$ *return*;
2. **for each** $x \in T$
3. **if** $\sup(H \cup \{x\}) = \sup(H)$ **let** $T := T - \{x\}$, $LC := LC \cup \{x\}$;
4. **if** $(H : LC,\ \sup(H))$ construct a new concept with $\sup(H)$
5. **for each** $p \in LC$ **do if** $p \succeq H$ **then** p is removed from LC;
6. **add** $(H : LC,\ \sup(H))$ to LMG;
7. **else** remove clutter; // see [6] for details
8. **for each** $x \in T$
9. **let** $H_x := H \cup \{x\}$ *and* $T_x := \{y \in T \mid y > x\}$;
10. **call** $DFS(H_x, T_x, LC)$;

Note that the definition above uses the θ-subsumption twice, for the selection of mingens and for the selection of SSMG.

Example 4. Table 2 shows formal concepts in a formal context $\mathbb{K}'_{\mathbf{fam}}$. SSMG for $D = (\mathsf{gf}(01)\mathsf{gf}(07)\mathsf{gf}(12), abcefgh)$ is g, fe, fh, and LMG for D is only fe. □

The Mining Algorithm The algorithm in Table 3 follows the depth-first search framework using a set-enumeration tree (SE-tree) [11]. A node v, including a head H and a tail T, has a search space for all itemsets $Z = H \cup T'$, where T' is a nonempty subset of T. For the node labelled by ab in the SE-tree for $\{a, b, c, d\}$, we have $H = ab$ and $T = cd$, and its search space consists of abc, abd and $abcd$. A local closure of H, $LC(H) = \{x \in H \cup T \mid H^I = Hx^I\}$, is a closure w.r.t ancestor nodes. For all ancestor nodes v' of v with head H' and tail T', $LC(H')$ is a proper subset of $LC(H)$. Hence H is considered as the local mingen for $LC(H)$.

6 Experimental Results and Conclusion

We have done experiments on two data sets and compared between the number of patterns, the first one was with $R_{\mathbf{fam}}$ in Fig. 3 and the latter was with

Table 4. Patterns on $R_{\mathbf{fam}}$.

$\sup_{\min}(\%)$	80	60	40	20
MAPIX	17	109	1063	4601
SSMG	12	21	39	-
LMG	6	13	22	-

Table 5. Patterns on Mutagenesis-Bonds.

$\sup_{\min}(\%)$	90	80	70	60	50	40	30	20	10
MAPIX	336	360	614	721	721	925	1467	2948	6630
SSMG	9	9	13	16	16	19	31	67	149
LMG	6	6	9	12	12	14	25	58	137

Mutagenesis-Bonds. Tables 4 and 5 show the number of patterns generated by MAPIX, SSMG, and LMG. In both data sets, though SSMG and LMG had large reduction of patterns compared with MAPIX, LMG reduces patterns even more than SSMG. Because of a complex structure of $R_{\mathbf{fam}}$, SSMG and LMG fault the computation with $\sup_{\min} = 20\%$.

We still need revise of the algorithm for scalability. We also need examine the efficacy in the intuitive sense, such as readability.

References

1. Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, 1998.
2. Jonas Poelmans, Paul Elzinga, Stijn Viaene, and Guido Dedene. Formal concept analysis in knowledge discovery: A survey. In *ICCS*, pp. 139–153, 2010.
3. Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT'1999*, Vol. 1540 of *LNCS*, pp. 398–416. Springer, 1999.
4. Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In *Discovery Science'2004*, Vol. 3245 of *LNCS*, pp. 16–31. Springer, 2004.
5. Yves Bastide, Nicolas Pasquier, Rafik Taouil, Gerd Stumme, and Lotfi Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. In *Computational Logic'2000*, Vol. 1861 of *LNCS*, pp. 972–986. Springer, 2000.
6. Guozhu Dong, Chunyu Jiang, Jian Pei, Jinyan Li, and Limsoon Wong. Mining succinct systems of minimal generators of formal concepts. In *DASFAA'2005*, Vol. 3453 of *LNCS*, pp. 175–187. Springer, 2005.
7. Saso Dzeroski. Multi-relational data mining: an introduction. *SIGKDD Explorations*, Vol. 5, No. 1, pp. 1–16, 2003.
8. Jun-ichi Motoyama, Shinpei Urazawa, Tomofumi Nakano, and Nobuhiro Inuzuka. A mining algorithm using property items extracted from sampled examples. In *ILP'2006*, Vol. 4455 of *LNCS*, pp. 335–350. Springer, 2007.
9. Yusuke Nakano and Nobuhiro Inuzuka. Multi-relational pattern mining based-on combination of properties with preserving their structure in examples. In *ILP'2010*, Vol. 6489 of *LNCS*, pp. 181–189. Springer, 2011.
10. Gerd Stumme. Iceberg query lattices for datalog. In *ICCS*, pp. 109–125, 2004.
11. Ron Rymon. Search through systematic set enumeration. In *KR'1992*, KR Proceedings, pp. 539–550. Morgan Kaufmann, 1992.

A Wordification Approach to Relational Data Mining: Early Results

Matic Perovšek[1,2], Anže Vavpetič[1,2], Nada Lavrač[1,2,3]

[1] Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia
[2] Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
[3] University of Nova Gorica, Nova Gorica, Slovenia
{matic.perovsek, anze.vavpetic, nada.lavrac}@ijs.si

Abstract. This paper describes a propositionalization technique called *wordification*. Wordification is inspired by text mining and can be seen as a transformation of a relational database into a corpus of documents. As in previous propositionalization methods, after the wordification step any propositional data mining algorithm can be applied. The most notable advantage of the presented technique is greater scalability - the propositionalization step is done in time linear to the number of attributes times the number of examples for one-to-many databases. Furthermore, wordification results in easily understandable propositional feature representation. We present our initial experiments on two real-life datasets.

Keywords: propositionalization, text mining, association rules

1 Introduction

Unlike traditional data mining algorithms, which look for models/patterns in a single table (propositional patterns), relational data mining algorithms look for models/patterns in multiple tables (relational patterns). For most types of propositional models/patterns there are corresponding relational patterns, for example: relational classification rules, relational regression tree, relational association rules, and so on.

For individual-centered relational databases, where there is a clear notion of individual, there exist techniques for transforming such a database into a propositional or single-table format. This transformation, called *propositionalization* [2, 4], can be used by traditional propositional learners, such as decision tree or classification rule learners.

In this paper we introduce a novel propositionalization technique called *wordification*. Wordification is inspired by text mining techniques and can be seen as a transformation of a relational database into a corpus of documents, where each document can be characerized by a set of properties describing the entries of a relational database.

Unlike other propositionalization techniques [2, 3, 4, 6], which search for good (and possibly complex) relational features to construct a subsequent propositional representation, this methodology focuses on a large number of simple

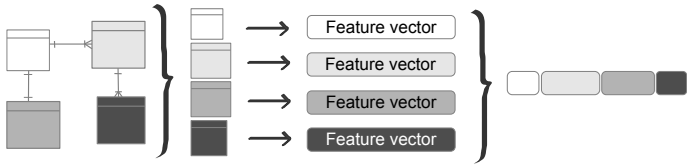


Fig. 1. Document (or feature vector) construction for one individual.

features with the aim of greater scalability. Since the feature construction step is very efficient, it can scale well for large relational databases. In fact, the presented methodology transforms a given database in time linear to the number of attributes times the number of examples for one-to-many databases. Furthermore, due to the simplicity of features, the generated features are easily interpretable by domain experts. On the other hand, this methodology suffers from the loss of information, since the generated features do not explicitly connect relations through variables. Instead, by using the Term Frequency–Inverse Document Frequency (TF-IDF) [1], it tries to capture the importance of a certain feature (attribute value) of a relation in an aggregate manner.

In this paper we report our initial experiments on two real-life relational databases: a collection of best and worst movies from the Internet Movie DataBase (IMBD) and a database of Slovenian traffic accidents.

The rest of the paper is organized as follows. In Section 2 we present the new wordification methodology. Section 3 presents the initial experiments and Section 4 concludes the paper by presenting some ideas for further work.

2 Wordification

This section presents the two main steps of the wordification propositionalization technique. First, a straightforward transformation from a relational database to a textual corpus is performed (Fig.1). One instance (i.e., one entry of the main table) of the initial relational database is transformed into one text document and the features (attribute values), describing the instance, constitute the words of this document. One document is constructed simply as a list of attribute-value pairs - words (or features) constructed as a combination of the table's name and the attribute's name with its discrete value:

$$[table\ name]\_[attribute\ name]\_[attributevalue].$$

Note that every attribute needs to be discretized beforehand in order to be able to represent every value as a word. For each instance, the features are first generated for the main table and then for each entry from the additional tables, and finally concatenated together.

Because we do not explicitly use existential variables in our features, we instead rely on the Term Frequency–Inverse Document Frequency (TF-IDF) measure to implicitly capture the importance of a certain feature for a given instance. In the context of text mining, TF-IDF reflects the representativeness of a certain word (feature) for the given document (instance). In the rest of this section we refer to instances as documents and to features as words.

Table 1. Example input for the standard East-West trains domain.

Train

tid	direction
1	east
2	west

Car

cid	tid	shape	roof	wheels
1	1	rectangle	none	2
2	1	rectangle	peaked	3
3	2	rectangle	none	2
4	2	hexagon	flat	2

Load

lid	cid	shape
1	1	rectangle
2	1	rectangle
3	2	circle
4	3	circle
5	4	circle
6	4	hexagon

1 [car_shape_rectangle, car_roof_none, car_wheels_2, load_shape_rectangle, load_shape_rectangle, car_shape_rectangle, car_roof_peaked, car_wheels_3, load_shape_circle] east
2 [car_shape_rectangle, car_roof_none, car_wheels_2, load_shape_circle, car_shape_hexagon, car_roof_flat, car_wheels_2, load_shape_circle, load_shape_hexagon] west

Fig. 2. The database from Table 1 in document representation.

For a given word w in a document d from corpus D, the TF-IDF is defined as follows: $\text{tfidf}(w,d) = \text{tf}(w,d) \times \log \frac{|D|}{|\{d \in D : w \in d\}|}$, where $tf(\cdot)$ represents the frequency of word w in document d. In other words, a certain word will have a high TF-IDF (i.e., the feature is given a high weight for this instance), if it is frequent within this document (instance) and infrequent in the given corpus (the original database). In other words, the weight of a word gives a strong indication of how relevant is the feature for the given individual. The TF-IDF weights can then be used either for filtering words with low importance or using them directly by the propositional learner.

The technique is illustrated on a simplified version of the well-known East-West trains domain, where the input database consists of two tables shown in Table 1; we have one east-bound and one west-bound train, each with two cars with certain properties. The `Train` table is the main table and the trains are the instances. We want to learn a classifier to determine the direction of an unseen train. For this purpose the class attribute (train direction) is not preprocessed and is only appended to the resulting feature vector of words.

First, the corresponding two documents (one for each train) are generated (Fig. 2). After this, the documents are transformed into a bag-of-words representation by calculating the TF-IDF values for each word of each document, with the class attribute column appended to the transformed bag-of-words table.

```
def wordification(table,individual)
    words=[]
    for att,val in table[individual]:
        words.append(table_att_val)
    #loop through tables which contain current table's foreign key
    for secondary_table in table.secondary_tables():
        words.extend(wordification(secondary_table,individual))
    return
#STEP1
documents={}
for individual in main_table:
    documents[individual]=wordification(main_table,individual)
#STEP2
feature_vectors=tfidf(documents)
results=propositional_learner(feature_vectors)
```

Fig. 3. Pseudo-code of the wordification algorithm.

Table 2. Table properties of the experimental data.

IMDB	#rows	#attributes
movies	166	4
roles	7,738	2
actors	7,118	4
movie_genres	408	2
movie_directors	180	2
directors	130	3
director_genres	243	3

Accidents	#rows	#attributes
accident	102,756	10
person	201,534	10

Table 3. Document properties after applying the wordification methodology.

Domain	Individual	#examples	#words	#words after filtering
IMDB	movie	166	7,453	3,234
Accidents	accident	102,759	186	79

3 Experimental results

This section presents the initial experiments of the wordification methodology. We performed association rule learning in combination with the wordification approach on two real-life datasets: the best and worst ranked IMDB movies database and the Slovenian traffic accidents database. Table 2 lists the characteristics of both databases.

The preprocessing procedure was performed on both databases as follows. First, the wordification step was applied as described in Section 2. Next, irrelevant features (which have the same value across all examples) were removed, resulting in less than half of the features (see Table 3). In order to prepare the data for association rule mining, the data was also binarized: a feature was assigned value *true* if the corresponding TF-IDF value was above 0.06, otherwise *false*.

IMDB database. The complete IMDB database is publicly available in the SQL format[1]. This database contains tables of movies, roles, actors, movie genres, directors, director genres.

The database used in our experiments consists only of the movies whose titles and years of production exist on IMDB's top 250 and bottom 100 chart[2]. The database therefore consists of 166 movies, along with all of their actors, genres and directors. Movies present in the IMDB's top 250 chart were added an additional label *goodMovie*, while those in the bottom 100 were marked as *badMovie*. Additionally, attribute age was discretized; a movie was marked as *old* if it was made before 1950, *fairlyNew* if it was produced between 1950 and 2000 and *new* otherwise.

After preprocessing the dataset using the wordification methodology, we performed association rule learning. Frequent itemsets were generated using RapidMiner's [5] FP-growth implementation. Next, association rules for the resulting

[1] `http://www.webstepbook.com/supplements/databases/imdb.sql`
[2] As of July 2, 2012

goodMovie ← director_genre_drama, movie_genre_thriller,
director_name_AlfredHitchcock. (Support: 5.38% Confidence: 100.00%)

movie_genre_drama ← goodMovie, actor_name_RobertDeNiro.
(Support: 3.59% Confidence: 100.00%)

director_name_AlfredHitchcock ← actor_name_AlfredHitchcock.
(Support: 4.79% Confidence: 100.00%)

director_name_StevenSpielberg ← goodMovie, movie_genre_adventure,
(Support: 1.79% Confidence: 100.00%) actor_name_TedGrossman.

Fig. 4. Examples of interesting association rules discovered in the IMDB database.

frequent itemsets were produced. Among all the discovered rules, several interesting rules were found. Figure 4 presents some of the interesting rules.

The first rule states that if the movie's genre is drama and is directed by Alfred Hitchcock, who is also known for drama movies, then the movie is a good movie. The second rule concludes that if a movie is good and Robert De Niro acts in it, than it must be a drama movie. The third interesting rule shows that Alfred Hitchcock acted only in the movies he also directed. The last rule implies that if Ted Grossman acts in a good adventure movie, then the director is Steven Spielberg (Ted Grossman usually plays the role of a stunt coordinator or performer).

Traffic accident database. The second dataset consists of all accidents that happened in Slovenia's capital city Ljubljana between the years 1995 and 2005. The data is publicly accessible from the national police department's website[3]. The database is multi-relational and consists of the information about the accidents along with all the accidents' participants.

noInjuries ← accident_trafficDensity_rare,
accident_location_parkingLot. (Support: 0.73% Confidence: 97.66%)

person_gender_male ← person_vehicleType_motorcycle.
(Support: 0.11% Confidence: 99.12%)

Fig. 5. Examples of interesting association rules discovered in the accidents database.

The data already contained discretized attributes, so further discretization was not needed. Similarly as with the IMDB databse, preprocessing using the wordification methodology, FP-growth itemset mining and association rule mining were performed. Figure 3 presents some of the interesting rules found in the Slovenian traffic accidents dataset.

The first rule indicates that if the traffic is rare and the accident happened in a parking lot, then no injuries occurred. The second rule implies that whenever a motorcycle is involved in an accident, a male person is involved.

[3] http://www.policija.si/index.php/statistika/prometna-varnost

4 Conclusion

This paper presented a novel propositionalization technique called wordification. This methodology is inspired by text mining and can be seen as a transformation of a relational database into a corpus of documents. As is typical for propositionalization methods, any propositional data mining algorithm can be applied after the wordification step. The most notable advantage of the presented technique is greater scalability - the propositionalization step is done in time linear to the number of attributes times the number of examples for one-to-many databases. Moreover, the methodology allows for producing simple, easy to understand features, and consequently, easily understandable rules.

We have presented initial results on two real-life databases: the best and worst movies from the IMDB database and a database of Slovenian traffic accidents. Given that we have found some interesting patterns using our methodology, we are motivated to further explore this approach on new datasets. In future work we will apply the methodology to larger databases to explore its potential limitations. Furthermore, we will experimentally compare this methodology with other known propositionalization techniques.

Acknowledgements

We are grateful to Marko Grobelnik and Peter Ljubič for their initial work on this topic. This work was supported by the Slovenian Research Agency and the FP7 European Commission projects MUSE (grant no. 296703) and FIRST (grant no. 257928).

References

[1] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[2] Stefan Kramer, Bernhard Pfahringer, and Christoph Helma. Stochastic propositionalization of non-determinate background knowledge. In *Proceedings of the Inductive Logic Programming Workshop*, pages 80–94. Springer, 1998.

[3] Ondřej Kuželka and Filip Železný. Block-wise construction of tree-like relational features with monotone reducibility and redundancy. *Machine Learning*, 83(2):163–192, 2011.

[4] Nada Lavrač, Sašo Džeroski, and Marko Grobelnik. Learning nonrecursive definitions of relations with LINUS. In *EWSL*, pages 265–281, 1991.

[5] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. YALE: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'06)*, pages 935–940, Philadelphia, PA, USA, 20-23 August 2006. ACM Press, NY, USA.

[6] Filip Železný and Nada Lavrač. Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, 62:33–63, 2006.

Hybrid Logical Bayesian Networks

Irma Ravkic, Jan Ramon, and Jesse Davis

Department of Computer Science
KU Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
irma.ravkic@cs.kuleuven.be
jan.ramon@cs.kuleuven.be
jesse.davis@cs.kuleuven.be

Abstract. Probabilistic logical models have proven to be very successful at modelling uncertain, complex relational data. Most current formalisms and implementations focus on modelling domains that only have discrete variables. Yet many real-world problems are hybrid and have both discrete and continuous variables. In this paper we focus on the Logical Bayesian Network (LBN) formalism. This paper discusses our work in progress in developing hybrid LBNs, which offer support for both discrete and continuous variables. We provide a brief sketch for basic parameter learning and inference algorithms for them.

1 Introduction

Real-world problems are hybrid in that they have discrete and continuous variables. Additionally, it is necessary to model the uncertain nature and complex structure inherent in these problems. Most existing formalisms cannot cope with all these challenges. Hybrid Bayesian networks [1] can model uncertainty about both discrete and continuous variables, but not relationships between objects in the domain. On the other hand, probabilistic logical models (PLM) [2–4] can model uncertainty in relational domains, but many formalisms restricted themselves to discrete data. More recently, several approaches have been proposed that augment PLMs in order to model hybrid relational domains. These include Adaptive Bayesian Logic Programs [5], ProbLog with continuous variables [6], and Hybrid Markov Logic Networks [7].

In this paper, we focus on upgrading another PLM framework called Logical Bayesian Networks [8] to model continuous variables. From our perspective, LBNs have several important advantages. One, they clearly distinguish the different components (i.e., the random variables, dependencies among the variables, and the CPDs of each variable) of a relational probabilistic model. Two, the CPDs are easily interpretable by humans, which is not the case in other formalisms, such as those based on Markov random fields. This paper reports on our preliminary work in progress on developing hybrid LBNs. We show how LBNs can naturally represent continuous variables. We also discuss a basic parameter learning algorithm and how Gibbs sampling can be used for inference.

2 Background

We briefly review Logical Bayesian Networks and Gibbs sampling.

2.1 Logical Bayesian Networks

A propositional Bayesian network (BN) compactly represents a probability distribution over a set of random variables $X = \{X_1, \ldots, X_n\}$. A BN is a directed, acyclic graph that contains a node for each variable $X_i \in X$. Each node in the graph has a conditional probability distribution $\theta_{X_i|Parents(X_i)}$ that gives the probability distribution over the values that a variable can take for each possible setting of its parents. A BN encodes the following probability distribution:

$$P_B(X_1, \ldots X_n) = \prod_{i=1}^{i=n} P(X_i|Parents(X_i)) \quad (1)$$

Logical Bayesian Networks upgrade propositional BNs to work with relational data [8]. LBNs contain four components: *random variable declarations*, *conditional dependencies*, *Conditional Probability Distributions (CPDs)* and a *logic program*. Semantically, a LBN induces a Bayesian network. Given a set of constants, the first two components of the LBN define the structure of the Bayesian network. The random variable declarations define which random variables appear in the network whereas conditional dependency relationships define the arcs that connect the nodes. Finally, the conditional probability functions determine the conditional probability distribution associated with each node in the network. We will illustrate each of these components using the well-known *university* example [9]. The logical predicates in this problem are `student/1`, `course/1`, and `takes/2`. Random variables start with capital letters and constants with lower-case letters. The logical predicates can then be used to define random variables as follows:

```
random(intelligence(S)):- student(S).
random(difficulty(C)):- course(C).
random(grade(S,C)):- takes(S,C).
random(ranking(S)):- student(S).
```

Conditional dependencies are represented by a set of clauses. The clauses state which variables depend on each other and determine which edges are included in a ground LBN. They take the following form:

```
grade(S,C) | intelligence(S),difficulty(C).
ranking(S) | grade(S,C) :- takes(S,C).
```

The last clause means that the ranking of a student depends on the grades of all courses the student takes. A CPD is associated with each conditional dependency in a LBN. In principle, any CPD is possible. However, LBNs typically

use logical probability trees. A logical probability tree is a binary tree where each internal node contains a logical test (conjunction of literals) and each leaf contains a probability distribution for a particular attribute. Examples are sorted down the tree based on whether they satisfy the logical test at an internal node.

The logic program component contains a set of facts and clauses that describes the background knowledge for a specific problem. It generates the ground Bayesian network. In the university example it may contain facts such as:

```
student(mary).
student(peter).
course(math).
takes(mary,math).
takes(peter,math).
```

The LBN specified in the running example induces a Bayesian network shown in Figure 1.

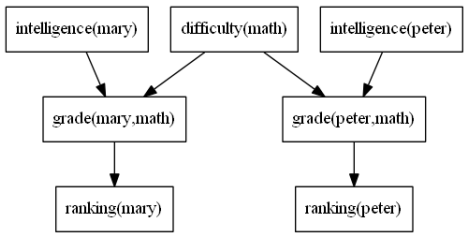


Fig. 1. Bayesian network induced by the simple *university* example

Notice that the logic program defines which random variables appear in the model, or in other words, it determines different interpretations (assignments to random variables).

2.2 Gibbs Sampling

Gibbs sampling is an instance of a Markov Chain Monte Carlo (MCMC) algorithm. It estimates a joint probability distribution over a set of random variables by simulating a sequence of draws from the distribution. It is commonly used when joint distributions over variables are not known or are complicated, but local dependency distributions are known and simple. To sample a value for a particular variable it is sufficient to only consider its Markov blanket. The time needed for Gibbs sampling to converge to a stationary distribution is dependent on the starting point and therefore in practice some number of examples are ignored (*burn-in* period). For more details see Casella and George, [10].

3 Our Approach

We now describe how we augment LBNs to model continuous variables.

3.1 Representation

It is relatively natural to incorporate continuous random variables in the LBNs. We do so by adding a new random variable declaration that indicates whether a variable is continuous and what its distribution is. For example, we could make the following declaration:

```
randomGaussian(numHours(C)):- course(C).
```

This states that `numHours(C)` is a Gaussian distributed continuous random variable if `C` is a course. Currently, we only allow Gaussian continuous variables, but it is straightforward to incorporate other distributions.
After being declared, continuous random variables can appear in conditional dependency clauses. For example:

```
numHours(C) | difficulty(C).
```

This clause states that the number of hours spent studying for a course `C` depends on the difficulty of the course. The difficulty of introducing continuous variables lies in a scenario when a discrete variable has continuous parents. Therefore, currently, we add a restriction that a discrete random variable cannot have continuous parents. This is a common restriction in hybrid BNs as well.

Logical CPDs can easily accommodate continuous variables by adding a Gaussian distribution in an appropriate leaf as in Figure 2. A Gaussian distribution with mean μ and standard deviation σ is:

$$N(\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{2}$$

3.2 Parameter Learning and Inference

When learning parameters we assume a known structure, that is, the structure of the probability tree is given. The examples are sets of interpretations where each interpretation refers to a particular instantiation of all random variables. We estimate the maximum likelihood of parameters. In the discrete case, this corresponds to a frequency of a specific variable value in a dataset. In the continuous case, this corresponds to computing the sample mean and standard deviation.

For estimating the mean and standard deviation we used a two-pass algorithm. It first computes the sample mean:

$$\mu = \frac{\sum_{i=1}^{n} y_i}{n} \tag{3}$$

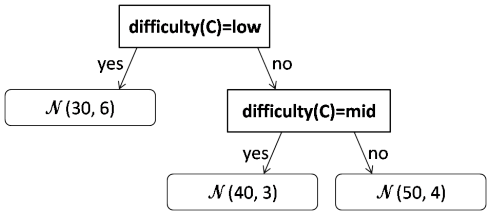


Fig. 2. Logical probability tree for `numHours(C)`

The standard deviation is calculated in the second pass through data by using:

$$\sigma = \frac{\sqrt{\sum_{i=1}^{n}(y_i - \mu)^2}}{n-1} \tag{4}$$

For inference, we have implemented Gibbs sampling which allows us to estimate the posterior probability of some variables given (a possibly empty) set of evidence variables. When querying continuous variables, we can answer several types of queries. We can estimate its mean value. Alternatively, we can estimate the probability that its value falls into some interval (i.e., estimate its cumulative distribution). We sample a continuous variable given its Markov blanket by generating a value from a Gaussian distribution given the appropriate mean and standard deviation coming from the CPD defined by its associated conditional dependency clause. A discrete variable having a continuous node as its child is sampled by using its Markov blanket, and the probability of a continuous child given its parents is computed using Equation (2).

4 Experiments

Currently, we have an implementation that works for small datasets. We have done preliminary experiments using synthetically generated data from the university domain that we have used as a running example in the paper. We augmented the task description with two continuous variables: `numHours/1` and `attendance/1`. The first one represents the number of hours a student spends studying for a particular course, and the second one denotes the number of hours students spent in class. We added two conditional dependency clauses making use of these variables:

```
numHours(C) | difficulty(C).
attendance(C) | satisfaction(S,C):-takes(S,C)
```

The first clause was described in Subsection 3.1 and the second clause states that a student is more likely to attend a class if (s)he enjoys the lectures.

To test the parameter learning, we generated synthetic datasets of varying sizes. Unsurprisingly, we found that we could learn accurate estimates of the

parameters. In terms of inference, we randomly selected some atoms as queries and some as evidence. On small examples, we found that the Gibbs sampler converged to the correct value after a reasonable number of iterations.

5 Conclusions

In this paper we presented a preliminary work on representation, learning and querying of hybrid logical Bayesian networks. Building on this preliminary work, in the future we will study other conditional probability models (e.g., using Poisson distributions), learning and inference in large-scale networks, and the application of hybrid LBNs in bio-medical applications.

Acknowledgements

This work is supported by the Research Fund K.U.Leuven (OT/11/051),

References

1. Murphy, K.: Inference and learning in hybrid Bayesian networks. University of California, Berkeley, Computer Science Division (1998)
2. Kersting, K., De Raedt, L.: 1 bayesian logic programming: Theory and tool. Statistical Relational Learning (2007) 291
3. De Raedt, L., Kimmig, A., Toivonen, H.: Problog: A probabilistic prolog and its application in link discovery. In: Proceedings of the 20th international joint conference on Artifical intelligence. (2007) 2468–2473
4. Richardson, M., Domingos, P.: Markov logic networks. Machine learning **62**(1) (2006) 107–136
5. Kersting, K., De Raedt, L.: Adaptive bayesian logic programs. Inductive Logic Programming (2001) 104–117
6. Gutmann, B., Jaeger, M., De Raedt, L.: Extending problog with continuous distributions. Inductive Logic Programming (2011) 76–91
7. Wang, J., Domingos, P.: Hybrid markov logic networks. In: Proceedings of the 23rd national conference on Artificial intelligence. Volume 2. (2008) 1106–1111
8. Fierens, D., Blockeel, H., Bruynooghe, M., Ramon, J.: Logical bayesian networks and their relation to other probabilistic logical models. Inductive Logic Programming (2005) 121–135
9. Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: International Joint Conference on Artificial Intelligence. Volume 16. (1999) 1300–1309
10. Casella, G., George, E.: Explaining the gibbs sampler. American Statistician (1992) 167–174

Towards an Automated Pattern Selection Procedure in Software Models

Alexander van den Berghe, Jan Van Haaren,
Stefan Van Baelen, Yolande Berbers, and Wouter Joosen

{firstname.lastname}@cs.kuleuven.be
iMinds-DistriNet, Department of Computer Science, KU Leuven
Celestijnenlaan 200A, 3001 Leuven, Belgium

Abstract. Software patterns are widely adopted to manage the rapidly increasing complexity of software. Despite their popularity, applying software patterns in a software model remains a time-consuming and error-prone manual task. In this paper, we argue that the relational nature of both software models and software patterns can be exploited to automate this cumbersome procedure. First, we propose a novel approach to selecting applicable software patterns, which requires only little interaction with a software developer. Second, we discuss how relational learning can be used to further automate this semi-automated approach.

Keywords: Relational Learning, Software Pattern Selection, Logic Programming, Application

1 Introduction

The complexity of both software and the software development process has increased rapidly over the past decades because of three reasons. The first reason is the steadily increasing complexity of the problems that software tackles. The second reason is the shift towards distributed software, which entails a number of additional issues to account for. The third reason is the relatively long lifetime of software, which is often much longer than that of the hardware it was originally developed for and which requires it to adapt to an ever changing environment.

Software patterns provide established solutions to recurring issues in software development [5, 1] and hence improve the overall quality, portability and readability of a software design. Although software patterns are widely adopted by software developers, applying patterns remains a mostly manual two-step task. First, a developer selects the most appropriate patterns based on his or her previous experiences. This is an increasingly difficult and time-consuming task

This research is partially funded by the Flemish government institution IWT (Institute for the Promotion of Innovation by Science and Technology in Flanders), by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund KU Leuven.

due to the steadily growing number of patterns. Second, a developer instantiates these patterns in the software design, which is a repetitive and error-prone task.

In this paper, we argue that the relational nature of both software models and software patterns can be exploited to automate this cumbersome procedure. We apply inductive logic programming techniques for representing these models and patterns as well as reasoning about them. First, we propose a semi-automated approach to selecting applicable patterns in a graphical software model. Our approach relies on a concise relational representation of both the available software patterns and a software model. The approach requires only little interaction with a software developer whereas current pattern selection procedures rely on an extensive specification of the design problem. Second, we briefly share ideas on how relational learning techniques can further automate our approach and hence reduce the amount of user interaction.

2 Background on Software Engineering

On a very high level, designing software is gathering requirements and ensuring these requirements are met in the final software system. Software developers typically identify a number of components, which each satisfy a subset of the requirements, in order to manage the complexity of software. Components offer their functionality through one or more interfaces via which they collaborate.

Software developers formally capture their design decisions in graphical models. Each model element can be annotated with additional information (e.g., implementation details) that is required during the development process. Although our approach is applicable to multiple types of software models, we restrict ourselves to the Component-and-Connector Model, which is one of the most important types of models during software development. Figure 1 shows an excerpt of a digital newspaper system, inspired by [10], in which clients can browse through news articles, read news articles and subscribe to specific categories of news.

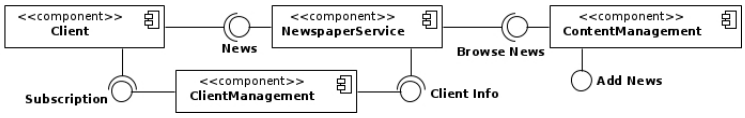


Fig. 1. Software model of a digital newspaper system where rectangles represent components and circles represent interfaces. A line denotes a component offering an interface whereas a line ending in half a circle denotes a component using an interface.

Software patterns offer established solutions to recurring design issues. Alongside a unique name, a software pattern comprises a generic description of both the design issue it addresses and the solution it proposes (i.e., independent of any concrete design and/or implementation decision). Furthermore, a software

pattern contains an overview of its consequences and trade-offs, which is helpful to decide on the most appropriate software pattern.

Automatically selecting applicable software patterns requires a formal representation of either the design issue a software pattern addresses or the solution it proposes. Our approach leverages Zdun and Avgeriou's concept of primitives [12] to formally represent a pattern's proposed solution. Primitives have precisely defined semantics and can be seen as building blocks for patterns. A software pattern combines several primitives to offer a solution to a specific design issue.

Figure 2 shows the `Client-Dispatcher-Server` pattern, which comprises the `Client`, `Dispatcher` and `Server` primitives [1].

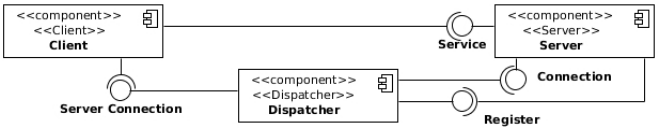


Fig. 2. Software model of the Client-Dispatcher-Server pattern.

3 Related Work

Pattern selection has only been given little attention in academic literature. Kampffmeyer and Zschaler [8] use a pattern intent ontology that can be queried with design issues and return all applicable patterns. Kim and El Khawand [9] first formally model each pattern as a set of pattern-specific roles and then determine which patterns are applicable by verifying which roles the software model fulfills. Hsueh et al. [7] introduce a goal-driven approach that proposes applicable patterns by asking relevant questions to the developer. Hasheminejad and Jalili [6] first classify the patterns and the design issue using text classification and then propose the best matching patterns from the design issue's class.

Current pattern selection techniques require either an extensive specification of the design problem (e.g., [8, 7, 6]) or the patterns' problem descriptions (e.g., [9]), which is often cumbersome and time-consuming, and considerably limits their usability and reliability. Besides, it is not obvious how to extend these techniques to new (types of) patterns since either an extensive analysis is required or the terminology in their description should be chosen carefully.

Although automated pattern selection has only been given little attention to date, the idea of tackling software engineering issues using machine learning is not new. One area of interest is that of software pattern detection, where the task is to identify applied software patterns when re-engineering legacy software. Correa et al. [2] perform this task using Prolog as a representation language. Another area of interest is that of software testing, where the task is to evaluate how well software systems meet their specification [3, 11].

4 Contributions

This section discusses the two main contributions of this paper. First, we propose Semi-Automated Role-Based Pattern Selection (SARBPS), a novel semi-automated approach for selecting applicable software patterns in software models. The approach exploits the relational nature of both software patterns and software models to significantly reduce the time required to decide which software patterns are applicable in a given software model. Second, we discuss how relational learning can be used to further automate our approach and hence reduce the amount of user interaction.

4.1 Contribution 1: Semi-Automated Role-Based Pattern Selection

We now discuss the three steps of the SARBPS approach in turn. The **first step** concerns representing the available software patterns as sets of primitives, where each primitive fulfills one or more roles. SARBPS represents this information as a knowledge base consisting of two classes of facts. The first class is the class of *entity facts* that enumerate the available patterns, primitives and roles. The second class is the class of *relation facts* that define relations between patterns and primitives on one hand and primitives and roles on the other hand. These relations express which primitives each pattern comprises and which role(s) each primitive fulfills. Roles are only meaningful if fulfilled by at least one primitive.

The **second step** involves annotating the components and interfaces of the software model with additional information. A software developer employs the requirements that the components and interfaces satisfy to assign one or multiple *roles*, which describe their properties and characteristics. Roles indicate which application-independent function the annotated element fulfills in a software system. For example, a component that provides services to other components can be assigned the `ServiceProvider` role. Only roles defined in the knowledge base constructed in the previous step can be assigned.

The **third step** involves selecting applicable patterns by establishing a mapping between the roles of the elements in the software model on one hand and the roles of the available patterns on the other hand. For each of the available patterns, SARBPS verifies whether it is applicable. A pattern is applicable if and only if each of the software model's roles is fulfilled by at least one of the pattern's primitives. SARBPS achieves this by querying the knowledge base it constructed in the first step using the roles assigned during the second step.

SARBPS uses two queries to retrieve all applicable patterns from the knowledge base. The first query returns for a given pattern and a given role which of the pattern's primitives fulfills this role or fails when no primitive is found. The second query returns a list of applicable patterns for a given set of roles by iteratively calling the first query for any possible combination of an available pattern and a given role.

Example 1. In order to illustrate our approach, we use the software model and software pattern shown in Figures 1 and 2 respectively. Due to space constraints, we only provide a high-level discussion in this paper.[2] In the **first step**, we add the `Client-Dispatcher-Server` pattern and the three primitives this pattern comprises, `Client`, `Dispatcher` and `Server`, to the knowledge base. We also add the roles `ServiceProvider` and `ServiceRequester`, which the `Server` and `Client` primitives fulfill respectively. In the **second step**, we assign the `ServiceProvider` role to both the `NewspaperService` and `ClientManagement` components, and the `ServiceRequester` role to the `Client` component. In the **third step**, querying the knowledge base for applicable software patterns returns the `Client-Dispatcher-Server` pattern.

Although our semi-automated pattern selection approach saves significant amounts of precious development time, manually assigning roles to components and interfaces is still an error-prone and time-consuming task, especially for large, real-world software systems. Therefore, we propose using relational learning to automate this task and share some ideas on how to do this in what follows.

4.2 Contribution 2: Towards Automated Role Annotations

When annotating a component or an interface with one or multiple roles, software developers leverage the descriptions of the requirements each component or interface fulfills. These descriptions are mostly free text but they tend to have a simple structure. For example, one of the requirements in our running example states that a journalist must be able to add an article to the system. In a medical system, a requirement could be that the medical staff must be able to add a new patient file to the system. Although the descriptions of these requirements are completely different, it is clear that they can be written as instances of the same abstract pattern since they share a common structure.

The missing building block is a convenient formal language for representing software requirements. Once we have such a language, we can naturally model requirements in a relational learning system and pose the assignment of roles to components and interfaces as a collective classification task. We can then represent components and interfaces as *entities*, and collaborations amongst these components and interfaces as *relations*. The classification task boils down to learning one or multiple roles for each component. The key idea is to leverage the expert knowledge and previous role labeling efforts of software engineers, who can possibly already assign roles to some of the components and interfaces.

The kernel-based relational learning framework kLog [4] is a suitable learning system to perform this task. An important advantage of kLog is that the outcome of its *graphicalization phase*, a ground entity-relationship model, is conceptually very similar to a software model and hence easily interpretable by a software engineer. Preliminary results on a toy example yield encouraging results.

[2] A thorough discussion of the running example and a Prolog implementation are available at `https://people.cs.kuleuven.be/alexander.vandenberghe/sarbps.html`.

5 Conclusion and Future Work

We have introduced a novel semi-automated approach for selecting applicable software patterns in software models, which makes developers to save significant amounts of development time. Our approach relies on a concise relational representation of both software patterns and software models, which allows reasoning about them and including additional software patterns in a straightforward way.

Our main research direction is further automating the approach using relational learning techniques. The key challenge is designing a convenient formal language for representing software requirements in a relational learning framework. Furthermore, we aim to limit the number of proposed patterns by incorporating each pattern's trade-offs and consequences.

Acknowledgments. We thank Paolo Frasconi and Luc De Raedt for providing access to the kLog implementation.

References

[1] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns. Wiley (1996)
[2] Correa, A., Werner, C., Zaverucha, G.: Object Oriented Design Expertise Reuse: an Approach Based on Heuristics, Design Patterns and Anti-Patterns. In: on Heuristics, Design Patterns and Anti-patterns, in Proceedings of the 6th International Conference on Software Reuse. pp. 336–352 (2000)
[3] DeMillo, R.A., Offutt, A.J.: Constraint-Based Automatic Test Data Generation. IEEE Trans. Softw. Eng. 17(9), 900–910 (Sep 1991), http://dx.doi.org/10.1109/32.92910
[4] Frasconi, P., Costa, F., De Raedt, L., De Grave, K.: kLog: A Language for Logical and Relational Learning with Kernels (2012), *arXiv:1205.3981v3*
[5] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional (1994)
[6] Hasheminejad, S.M.H., Jalili, S.: Design Patterns Selection: An Automatic Two-Phase Method. The Journal of Systems and Software 85(2), 408–424 (Feb 2012)
[7] Hsueh, N.L., Kuo, J.Y., Lin, C.C.: Object-Oriented Design: A Goal-driven and Pattern-based Approach. Software and Systems Modeling 8(1), 67–84 (2009)
[8] Kampffmeyer, H., Zschaler, S.: Finding the Pattern You Need: The Design Pattern Intent Ontology. In: Model Driven Engineering Languages and Systems. pp. 211–225. No. 4735 in Lecture Notes in Computer Science (2007)
[9] Kim, D.K., El Khawand, C.: An Approach to Precisely Specifying the Problem Domain of Design Patterns. Journal of Visual Languages & Computing 18(6), 560–591 (2007)
[10] Van Landuyt, D., Op de beeck, S., Truyen, E., Verbaeten, P.: Building a Digital Publishing Platform Using AOSD. In: LNCS Transactions on Aspect-Oriented Software Development. vol. 9, pp. 1–34 (December 2010)
[11] Vanmali, M., Last, M., Kandel, A.: Using a neural network in the software testing process. International Journal of Intelligent Systems 17(1), 45–62 (2002), http://dx.doi.org/10.1002/int.1002
[12] Zdun, U., Avgeriou, P.: Modeling Architectural Patterns Using Architectural Primitives. ACM SIGPLAN Notices 40(10), 133–146 (Oct 2005)

Non-monotone Dualization via Monotone Dualization

Yoshitaka Yamamoto[1], Koji Iwanuma[1], Katsumi Inoue[2]

[1] University of Yamanashi
4-3-11 Takeda, Kofu-shi, Yamanashi 400-8511, Japan.
[2] National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan.

1 Introduction

The *non-monotone dualization* (NMD) is one of the most essential tasks required for finding hypotheses in various ILP settings, like *learning from entailment* [1, 2] and *learning from interpretations* [3]. Its task is to generate an irredundant prime CNF formula ψ of the dual f^d where f is a *general* Boolean function represented by CNF [4]. The DNF formula ϕ of f^d is easily obtained by De Morgan's laws interchanging the connectives of the CNF formula. Hence, the main task of NMD is to translate the DNF ϕ to an equivalent CNF ψ. This translation is used to find an alternative representation of the input form. For instance, given a set of models, it is desirable to seek underlying structure behind the models.

Example 1. Suppose that a set of models M are given as follows:

$$M = \{\ (bird \wedge normal \wedge flies),\ (\neg flies \wedge \neg normal),\ (\neg flies \wedge \neg bird)\ \}.$$

By treating M as the DNF formula, we translate it into CNF with NMD:

$$H = (bird \vee \neg flies) \wedge (normal \vee \neg flies) \wedge (flies \vee \neg normal \vee \neg bird).$$

In fact, H is regarded as a hypothesis in learning from interpretations [3].

In contrast, by converting a given CNF formula into DNF, we obtain the models satisfying the CNF formula. This fact shows an easy reduction from SAT problems to NMD, and then gives NP-hardness of it [5]. In this context, the research has been much focused on some restricted classes of NMD.

Monotone dualization (MD) is one such class that deals with *monotone* Boolean functions for which CNF formulas are negation-free [6, 7]. MD is one of the few problems whose tractability status with respect to polynomial-total time is still unknown. Besides, it is known that MD has many equivalent problems in discrete mathematics, such as the minimal hitting set enumeration and the transversal hypergraph computation [6]. Thus, this class has received much attention that yields remarkable algorithms: in terms of complexity, Fredman and Khachiyan [8] show that MD is solvable in a quasi-polynomial-total time (i.e.,

$(n+m)^{O(log(n+m))}$ where n and m denote the input and output size, respectively). Uno [9] shows a practically fast algorithm whose average computation time is experimentally $O(n)$ per output, for randomly generated instances.

This paper aims at clarifying whether or not NMD can be solved using these techniques of MD, and if it can be then how it is realized. In general, it is not straightforward to use them because of the following two problems in NMD:

- NMD has to treat *redundant clauses* like resolvents and tautologies.

  Example 2. Let a CNF formula ϕ be $(x_1 \vee x_2) \wedge (\overline{x_2} \vee x_3)$. If we treat negated variables as regular variables, we can apply MD to ϕ and obtain the CNF formula $\psi = (x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_3) \wedge (x_2 \vee \overline{x_2}) \wedge (x_2 \vee x_3)$. However, ψ contains the tautology $x_2 \vee \overline{x_2}$ and the resolvent $x_1 \vee x_3$ of $x_1 \vee \overline{x_2}$ and $x_2 \vee x_3$.

- Unlike MD, the output of NMD is *not* necessarily *unique*. It is known that the output of MD uniquely corresponds to the set of all the prime implicates of f^d [10]. In contrast, some prime implicates can be redundant in NMD problems. Thus, the output of NMD corresponds to an irredundant subset of the prime implicates. However, such a subset is not unique in general.

For the first problem, this paper shows a technique to prohibit any resolvents from being generated in MD. This is done by simply adding some tautologies to the input CNF formula ϕ in advance. We denote by ϕ_t and ψ_t the extended input formula and its output by MD, respectively. Then, ψ_t contains no resolvents.

Example 3. Recall Example 2. We consider the CNF formula $\phi_t = (x_1 \vee x_2) \wedge (\overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_2})$ obtained by adding one tautology $x_2 \vee \overline{x_2}$. Then, MD generates the CNF formula $\psi_t = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee \overline{x_2}) \wedge (x_2 \vee x_3)$. Indeed, ψ_t does not contain the resolvent $x_1 \vee x_3$, unlike ψ in Example 2.

By removing all tautologies from ψ_t, we obtain an irredundant CNF formula, denoted by ψ_{ir}. Note that ψ_{ir} is $(x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_3)$ in Example 3.

We next address the second problem using a good property of ψ_{ir}: a subset of the prime implicates is irredundant (i.e., an output of NMD) if and only if it subsumes ψ_{ir} but never subsumes ψ_{ir} if any clause is removed from it. This particular relation is called *minimal subsumption*. We then show that every subset satisfying the minimal subsumption is generated by MD. In this way, we reduce an original NMD problem into two MD problems: the one for computing ψ_{ir}, and the other for generating a subset corresponding to an output of NMD. Due to space limitations, full proofs are omitted.

2 Background

2.1 Preliminaries

A *Boolean function* is a mapping $f : \{0,1\}^n \to \{0,1\}$. We write $g \models f$ if f and g satisfy $g(v) \leq f(v)$ for all $v \in \{0,1\}^n$. g is (*logically*) *equivalent* to f, denoted by $g \equiv f$, if $g \models f$ and $f \models g$. A function f is *monotone* if $v \leq w$

implies $f(v) \leq f(w)$ for all $v, w \in \{0,1\}^n$; otherwise it is *non-monotone*. Boolean variables $x_1, x_2, \ldots, x_n$ and their negations $\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}$ are called *literals*. The *dual* of a function f, denoted by f^d, is defined as $\overline{f}(\overline{x})$ where $\overline{f}$ and $\overline{x}$ is the negation of f and x, respectively.

A *clause* (*resp. term*) is a disjunction (*resp.* conjunction) of literals which is often identified with the set of its literals. It is known that a clause is *tautology* if it contains complementary literals. A clause C is an *implicate* of a function f if $f \models C$. An implicate C is *prime* if there is no implicate C' such that $C' \subset C$.

A *conjunctive normal form* (CNF) (*resp. disjunctive normal form* (DNF)) formula is a conjunction of clauses (*resp.* disjunction of terms) which is often identified with the set of clauses in it. A CNF formula ϕ is *irredundant* if $\phi \not\equiv \phi - \{C\}$ for every clause C in ϕ; otherwise it is *redundant*. ϕ is *prime* if every clause in ϕ is a prime implicate of ϕ; otherwise it is *non-prime*. Let ϕ_1 and ϕ_2 be two CNF formulas. ϕ_1 *subsumes* ϕ_2, denoted by $\phi_1 \succeq \phi_2$, if there is a clause $C \in \phi_1$ such that $C \subseteq D$ for every clause $D \in \phi_2$. In turn, ϕ_1 *minimally subsumes* ϕ_2, denoted by $\phi_1 \succeq^{\natural} \phi_2$, if ϕ_1 subsumes ϕ_2 but $\phi_1 - \{C\}$ does not subsume ϕ_2 for every clause $C \in \phi_1$.

Let ϕ be a CNF formula. $\tau(\phi)$ denotes the CNF formula obtained by removing all tautologies from ϕ. We say ϕ is *tautology-free* if $\phi = \tau(\phi)$. Now, we formally define the dualization problem as follows.

Definition 1 (Dualization problem).

Input:	A tautology-free CNF formula ϕ
Output:	An irredundant prime CNF formula ψ such that ψ is logically equivalent to ϕ^d

We call it *monotone dualization* (MD) if ϕ is negation-free; otherwise it is called *non-monotone dualization* (NMD). As well known [6], the task of MD is equivalent to enumerating the *minimal hitting sets* (MHSs) of a family of sets.

2.2 MD as MHS enumeration

Definition 2 ((Minimal) Hitting set). Let Π be a finite set and $\mathcal{F}$ be a subset family of Π. A finite set E is a *hitting set* of $\mathcal{F}$ if for every $F \in \mathcal{F}$, $E \cap F \neq \emptyset$. A finite set E is a *minimal hitting set* (MHS) of $\mathcal{F}$ if E satisfies that

1. E is a hitting set of $\mathcal{F}$;
2. For every subset $E' \subseteq E$, if E' is a hitting set of $\mathcal{F}$, then $E' = E$.

Note that any CNF formula ϕ can be identified with the family of clauses in ϕ. Now, we consider the CNF formula, denoted by $M(\phi)$, which is the conjunction of all the MHSs of the family ϕ. Then, the following holds.

Theorem 1. [10] Let ϕ be a tautology-free CNF formula. A clause C is in $\tau(M(\phi))$ if and only if C is a non-tautological prime implicate of ϕ^d.

Hence, the output of MD for ϕ uniquely corresponds to $\tau(M(\phi))$: the set of all MHSs of the family ϕ by Theorem 1.

2.3 NMD as MHS enumeration

Our motivation is to clarify whether or not any NMD problem can be reduced into some MD problems. While MD is done by the state-of-the-art algorithms to compute MHSs [9], it is not straightforward to use them for NMD. Here, we review the two problems explained before in the context of MHS enumeration.

1. *Appearance of redundant clauses*: $\tau(M(\phi))$ is prime but not irredundant.

Example 4. Recall the input CNF formula $\phi_2 = \{\{x_1,\ x_2\}, \{\overline{x_2},\ x_3\}\}$ of Example 2. Then, $\tau(M(\phi_2)) = \{\{x_1,\ \overline{x_2}\},\ \{x_1,\ x_3\},\ \{x_2,\ x_3\}\}$. Indeed, this contains the redundant clause $\{x_1,\ x_3\}$.

2. *Non-uniqueness of NMD solutions*: there are many subsets of $\tau(M(\phi))$ that are prime and irredundant.

Example 5. Let the input CNF formula ϕ be $\{\{x_1,\ \overline{x_2},\ \overline{x_3}\},\ \{\overline{x_1},\ x_2,\ x_3\}\}$. $\tau(M(\phi))$ consists of the non-tautological prime implicates as follows:

$$\tau(M(\phi)) = \{\{x_1,\ x_2\},\ \{\overline{x_1},\ \overline{x_3}\},\ \{\overline{x_2},\ x_3\},\ \{x_1,\ x_3\},\ \{\overline{x_1},\ \overline{x_2}\},\ \{\overline{x_3},\ x_2\}\}.$$

Then, we may notice that there are at least two irredundant subsets of $\tau(M(\phi))$:

$$\psi_1 = \{\{x_1,\ x_2\},\ \{\overline{x_1},\ \overline{x_3}\},\ \{\overline{x_2},\ x_3\}\}.\ \psi_2 = \{\{x_1,\ x_3\},\ \{\overline{x_1},\ \overline{x_2}\},\ \{\overline{x_3},\ x_2\}\}.$$

Indeed, ψ_1 is equivalent to ψ_2, and thus both are equivalent to $\tau(M(\phi))$.

To address the two problems, this paper focuses on the following CNF formula.

Definition 3 (Bottom formula). Let ϕ be a tautology-free CNF formula and $Taut(\phi)$ be $\{\ x \vee \overline{x}\ \mid\ \phi$ *contains both* x *and* $\overline{x}\ \}$. Then, the *bottom formula wrt* ϕ (in short, bottom formula) is defined as the CNF formula $\tau(M(\phi \cup Taut(\phi)))$.

3 Properties of bottom formulas

Now, we show two properties of bottom formulas.

Theorem 2. Let ϕ be a tautology-free CNF formula. Then, the bottom formula wrt ϕ is irredundant.

Example 6. Recall ϕ_2 in Example 4. Then, $Taut(\phi_2)$ is $\{x_2 \vee \overline{x_2}\}$, and the bottom formula wrt ϕ_2 is $\{\{x_1,\ \overline{x_2}\},\ \{x_2,\ x_3\}\}$. Indeed, it is irredundant, since it does not contain the resolvent $\{x_1,\ x_3\}$, unlike Example 4.

While any bottom formula is irredundant, it is not necessarily prime.

Example 7. Recall the CNF formula ϕ in Example 5. Since $Taut(\phi)$ is the set $\{\{x_1,\ \overline{x_1}\},\ \{x_2,\ \overline{x_2}\},\ \{x_3,\ \overline{x_3}\}\}$, the bottom formula is as follows:

$$\{\{x_1, x_2, x_3\},\ \{\overline{x_3}, x_2, x_1\},\ \{\overline{x_3}, x_2, \overline{x_1}\},\ \{\overline{x_2}, x_3, x_1\},\ \{\overline{x_2}, x_3, \overline{x_1}\},\ \{\overline{x_2}, \overline{x_3}, \overline{x_1}\}\}.$$

We write $C_1, C_2, \ldots, C_6$ for the above clauses in turn (i.e., C_4 is $\{\overline{x_2},\ x_3,\ x_1\}$). We then notice that the bottom formula is non-prime, because it contains a non-prime implicate C_1 whose subset $\{x_1,\ x_2\}$ is an implicate of ϕ^d.

As shown in Example 7, the bottom formula itself is not necessarily an output of NMD. However, every NMD output is logically described with this formula.

Theorem 3. Let ϕ be a tautology-free CNF formula. ψ is an output of NMD for ϕ iff $\psi \subseteq \tau(M(\phi))$ and ψ minimally subsumes the bottom formula wrt ϕ.

Example 8. Recall Example 5 and Example 7. Fig. 1 describes the subsumption lattice bounded by two irredundant prime outputs ψ_1 and ψ_2 as well as the bottom formula $\{C_1, C_2, \ldots, C_6\}$. The solid (*resp.* dotted) lines show the subsumption relation between ψ_1 (*resp.* ψ_2) and the bottom formula. We then notice that both outputs ψ_1 and ψ_2 minimally subsume the bottom formula.

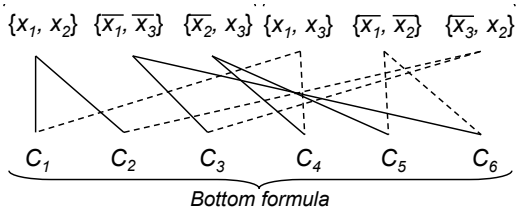


Fig. 1. Subsumption lattice bounded by NMD outputs and the bottom formula

4 Reconstructing NMD into MD

Theorem 3 shows that every NMD output ψ can be generated by selecting a subset ψ of $\tau(M(\phi))$ that minimally subsumes the bottom formula. Now, we show that the task of this selection is done by MD computation.

Let the bottom formula be $\{C_1,\ C_2,\ \ldots,\ C_n\}$. We then denote by S_i $(1 \leq i \leq n)$ the set of clauses in $\tau(M(\phi))$ each of which is a subset of C_i. $\mathcal{F}_\phi$ denotes the family of those sets $\{S_1, S_2, \ldots, S_n\}$.

Theorem 4. Let ϕ be a tautology-free CNF formula. ψ is an output of NMD for ϕ if and only if ψ is an MHS of $\mathcal{F}_\phi$.

Example 9. Recall Example 8. We denote each clause of ψ_1 and ψ_2 in Fig. 4 by $D_1, \ldots, D_6$, starting from left to right (i.e., D_4 is $\{x_1,\ x_3\}$). $\mathcal{F}_\phi$ is as follows:

$$\mathcal{F}_\phi = \{\{D_1,\ D_4\},\ \{D_1,\ D_6\},\ \{D_2,\ D_6\},\ \{D_3,\ D_4\},\ \{D_3,\ D_5\},\ \{D_2,\ D_5\}\}.$$

By MHS computation, we have the five solutions, which contain $\{D_1,\ D_2,\ D_3\}$ and $\{D_4,\ D_5,\ D_6\}$ that correspond to ψ_1 and ψ_2, respectively.

5 Concluding remarks

This paper has presented a reduction technique from arbitrary NMD problem to two equivalent MD problems. Whereas algorithms and computation on MD has been extensively studied, it was not clarified whether or not NMD can be

solved using the state-of-the-art MD computation. For this open problem, we give a solution how it is to be realized.

Our result can be used to investigate the complexity of NMD from the viewpoint of MD computation. For instance, the complexity of generating one NMD output can be described as follows:

$$(n+t+x)^{O(log(n+t+x))} + (x+1)^{O(log(x+1))}, \quad (1)$$

where n, t and x are the sizes of the input formula ϕ, the tautologies $Taut(\phi)$ and the bottom formula $\tau(M(\phi \cup Taut(\phi)))$, respectively. This is simply derived from the result on the complexity of MD computation [8]. Note that the right-hand term in Formula (1) comes from the complexity of the *incremental* MHS generation problem [11]. In contrast, the complexity of MD for computing the prime implicates of ϕ is described as $(n+m)^{O(log(n+m))}$ where m is the size of $\tau(M(\phi))$. Hence, the difference of their complexities lies in how big the blow-up in size can be when the bottom formula is derived. In other words, our proposal is not a polynomial reduction to MD, and does not establish NP-completeness of MD. However, if x is equal to m, like Example 4 and Example 7, there is no difference between NMD and MD in terms of the complexity. In this sense, it is an important future work to characterize those subclasses of NMD with only polynomial increase in the problem size x.

References

1. Inoue, K.: Induction as consequence finding. Machine Learning **55(2)** (2004) 109–135
2. Yamamoto, Y., Inoue, K., Iwanuma, K.: Inverse subsumption for complete explanatory induction. Machine Learning **86(1)** (2011) 115–139
3. DeRaedt, L.: Logical settings for concept-learning. Artificial Intelligence **95** (1997) 187–201
4. Eiter, T., Makino, K.: Abduction and the dualization problem. In: Proceedings of the 6th Int. Conf. on Discovery Science. Volume 2843 of LNCS. (2003) 1–20
5. Eiter, T., Ibaraki, T., Makino, K.: Recognition and dualization of disguised bidual Horn functions. Information Processing Letters **82** (2002) 283–291
6. Eiter, T., Makino, K., Gottlob, G.: Computational aspects of monotone dualization: A brief survey. Discrete Applied Mathematics **156** (2008) 2035–2049
7. Hagen, M.: Algorithmic and computational complexity issues of MONET. PhD thesis, Friedrich Schiller University Jena (2008)
8. Fredman, M., Khachiyan, L.: On the complexity of dualization of monotone disjunctive normal forms. Algorithms **21** (1996) 618–628
9. Uno, T.: A practical fast algorithm for enumerating minimal set coverings. In: Proceedings of the 83rd SIGAL Conf. of the Information Processing Society of Japan. (2002) 9–16
10. Rymon, R.: An SE-tree based prime implicant generation algorithm. Annals of Mathematics and Artificial Intelligence (1994) 351–366
11. Elbassioni, K.M.: On the complexity of monotone dualization and generating minimal hypergraph transversals. Discrete Applied Mathematics **156** (2008) 2109–2123