

FAST CONSTRUCTION OF RELATIONAL FEATURES FOR MACHINE LEARNING

ONDŘEJ KUŽELKA

Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

A thesis submitted for the degree of Doctor of Philosophy (Ph.D.)
Study Programme No. P2612-Electrotechnics and Informatics
Branch No. 3902V035 - Artificial Intelligence and Biocybernetics

May 2013

SUPERVISOR:
Doc. Ing. Filip Železný, PhD.

Ondřej Kuželka: *Fast Construction of Relational Features for Machine Learning*, A thesis submitted
for the degree of Doctor of Philosophy (Ph.D.)
Study Programme No. P2612-Electrotechnics and Informatics
Branch No. 3902V035 - Artificial Intelligence and Biocybernetics, © May 2013

ABSTRACT

This thesis deals with the problem of making construction of relational features more efficient. Specifically, it focuses on the situations when we know or assume that a good set of features can be constructed from features having some special restricted form. The thesis introduces novel algorithms for fast construction of relational features and several other methods and ideas applicable in relational learning in general. The presented feature-construction algorithms can be categorized into two groups: algorithms based on a block-wise strategy exploiting monotonicity of redundancy and reducibility and algorithms based on a new notion of bounded least general generalization.

The main idea underlying the first type of algorithms is that *redundancy* and *reducibility* of features can be made monotonous in a certain *block-wise feature-construction strategy* which enables a quick pruning of the set of features that need to be constructed. The introduced feature-construction algorithms RelF, HiFi and Poly are able to construct exhaustive sets of relatively long non-redundant *treelike* features. The new algorithms are shown to be able to obtain better predictive accuracies than state-of-the-art relational learning systems in many domains.

The main idea of the second type of algorithms presented in this thesis is to weaken existing notions of θ -*subsumption*, θ -*reduction* and *least general generalization* and parametrize them. The experiments that we performed indicate that the new algorithms are able to achieve state-of-the-art accuracies using only small sets of very long complex features.

Besides the main contributions which are the feature-construction algorithms, this thesis also studies so-called polynomial features which are multivariate aggregation features suitable for learning in hybrid domains. Another smaller contribution presented in this thesis is the introduction of the concept of a safe reduction of a learning example and the development of methods for its utilization as a preprocessing technique. Using the idea of safe reduction, we are able to speed up existing relational learning algorithms by preprocessing their input.

We also describe the use of some of the presented algorithms as components of bioinformatics methods for solving the problems of the DNA-binding propensity prediction and the antimicrobial activity prediction of peptides.

ABSTRAKT

Tato práce se zabývá otázkou jak zefektivnit proces konstrukce relačních rysů. Konkrétně se zaměřuje na problémy, kdy víme nebo předpokládáme, že je možné zkonstruovat *dobrou* množinu relačních rysů pouze z rysů z nějaké omezené třídy. Práce představuje nové algoritmy pro rychlou konstrukci relačních rysů a několik dalších metod a myšlenek použitelných v relačním strojovém učení obecně.

Představené algoritmy lze rozdělit do dvou základních skupin: na algoritmy založené na strategii skládání *stavebních bloků* využívající monotónnosti *redundance* a *redukovatelnosti* rysů a na algoritmy založené na nově zavedeném pojmu *omezené nejmenší společné zobecnění*. Hlavní myšlenka algoritmů z první skupiny je založena na poznatku, že *redundance* a *redukovatelnost* relačních rysů může být v jistém typu strategií založených na skládání *stavebních bloků* monotónní, což umožňuje těmto algoritmům rychle prořezávat množinu rysů, které je potřeba zkonstruovat. Představené algoritmy RelF, HiFi a Poly jsou schopné rychle konstruovat neredundantní množiny poměrně velkých relačních rysů se *stromovou strukturou*. Tyto algoritmy jsou schopné dosáhnout vyšších prediktivních přesností než jiné existující systémy relačního strojového učení v mnoha doménách. Základní myšlenkou, na které jsou založeny algoritmy z druhé skupiny, je zobecnění a parametrizace existujících pojmů θ -subsumpce, θ -redukce a nejmenšího společného zobecnění. Experimenty provedené s tímto typem algoritmů ukazují, že jsou tyto algoritmy schopné dosahovat přesností stejných nebo vyšších než existující algoritmy, a to i jen pomocí malých množin velkých relačních rysů.

Kromě hlavních výsledků, jimiž jsou představované algoritmy pro konstrukci relačních rysů, studuje tato práce také tzv. polynomiální relační rysy založené na vícerozměrné relační agregaci, které jsou vhodné především pro hybridní domény. Dalším menším přínosem této práce je zavedení pojmu *bezpečné redukce* trénovacích příkladů a metod pro využití bezpečné redukce k předzpracování trénovacích dat. Pomocí této metody jsme schopni zrychlit existující systémy relačního učení.

Kromě výše uvedeného uvádíme v této práci také způsoby využití představených algoritmů v rámci složitějších systémů pro predikování schopnosti proteinů vázat se na DNA a pro predikci antimikrobiální aktivity peptidů.

*All right, brain. You don't like me and I don't like you,
but let's just do this and I can get back to killing you with beer.*

— Homer Simpson

ACKNOWLEDGEMENTS

I would like to thank my advisor *Doc. Ing. Filip Železný, Ph.D.* for introducing me to the area of relational learning and for his guidance which was as good as \aleph_1 . I would also like to thank my colleague *Andrea Szabóová* for the great collaboration which was as great as 2^{\aleph_0} ... and hey, in this acknowledgement I assume the continuum hypothesis to be false.

I would also like to thank *Matěj Holec, Prof. RNDr. Roman Barták, Ph.D.* and *Prof. Jakub Tolar, M.D., Ph.D.* Thanks also go to the *Ticos* who helped us with developing a user interface for some of our algorithms: *Laura Vásquez, Keylor Chaves, Beatriz González* and *Hugo Trejos*.

My warm thanks go to *my family* for their love and support and also to *my friends*.

I acknowledge support by the following grants provided by the Czech Science Foundation: P202/12/2032 (2012-2013), 103/11/2170 (2011-2012), and 201/08/0509 (2008-2010), and by the following grants provided by the Grant Agency of the Czech Technical University in Prague, grants SGS10/073/OHK3/1T/13 (2010) and SGS11/155/OHK3/3T/13 (2011-2013).

CONTENTS

1	INTRODUCTION	1
1.1	Problem Statement	1
1.2	Main Contributions	2
1.3	Thesis Organization	2
I	BACKGROUND	3
2	RELATIONAL MACHINE LEARNING	4
2.1	Logic-Based Relational Machine Learning	4
2.2	Logic-based Relational Learning Based on Propositionalization	6
2.3	Statistical Relational Learning	7
2.4	Function-Freeness Assumption	8
3	SUBSUMPTION AND CONSTRAINT SATISFACTION	9
3.1	θ -subsumption and the Subsumption Theorem	9
3.2	θ -subsumption as a Constraint Satisfaction Problem	10
3.3	Tree Decompositions and Treewidth	11
4	THE CONCEPT OF A RELATIONAL FEATURE	14
4.1	Notation	14
4.2	A Simple Probabilistic Framework	15
4.3	Boolean Features	16
4.4	Counting Features	18
4.5	Polynomial Features	19
4.6	Generalized Multivariate-Aggregation Features	22
4.7	Related Work	22
4.8	Conclusions	23
4.9	Proofs	24
II	BLOCK-WISE CONSTRUCTION OF RELATIONAL FEATURES: RELF, HIFI AND POLY	26
5	RELF AND HIFI	27
5.1	τ -Features	28
5.2	Treelike Structure of τ -Features	30
5.3	Blocks	31
5.4	Reducibility	32
5.5	A Fast Algorithm for H-reduction	34
5.6	Extensions and Domains	35
5.7	Redundancy	36
5.8	Feature Construction Algorithm RELF	39
5.9	Feature Construction Algorithm HiFi	41
5.10	Experiments	43
5.10.1	Datasets	43
5.10.2	Feature Construction	44
5.10.3	Classification	45
5.11	Related Work	47
5.12	Conclusions	49
5.13	Proofs	49
5.14	Notes on Complexity of Template-Based Feature Construction	53
5.14.1	Negative Results	53
5.14.2	Positive Results	55
6	POLY	56
6.1	Evaluation of Polynomial Features	56

6.2	Evaluation of Generalized Multivariate Aggregation Features	58
6.3	Redundancy of Polynomial Features	59
6.4	Redundancy of Generalized Multivariate Aggregation Features	60
6.5	Construction of Features	60
6.6	Experiments	61
6.6.1	Polynomial Features for Propositionalization	61
6.6.2	Polynomial Features for Construction of Gene Sets	62
6.7	Conclusions	64
6.8	Proofs	64
III	RELATIONAL LEARNING MODULO HYPOTHESIS LANGUAGES	65
7	RELATIONAL LEARNING WITH BOUNDED LGG	66
7.1	Preliminaries: LGG, k-consistency, Generalized Arc Consistency	67
7.2	Bounded Subsumption	69
7.3	Bounded Reduction	71
7.4	Bounded Least General Generalization	73
7.5	Learning with Bounded Least General Generalization	76
7.6	Instantiations of the Framework	79
7.6.1	Clauses of Bounded Treewidth	79
7.6.2	Acyclic and Treelike Clauses	80
7.6.3	Clauses Given Implicitly by a Filtering Algorithm	82
7.6.4	Clauses Constrained by a User-definable Language Bias	82
7.7	Experiments	84
7.7.1	Implementation	84
7.7.2	Datasets	84
7.7.3	Bounded Reductions	85
7.7.4	Comparison with a Baseline Bottom-up Learning Algorithm	86
7.7.5	Comparison with Existing Relational Learning Algorithms	86
7.8	Related Work	87
7.9	Conclusions	90
7.10	Propositions and Proofs	90
7.11	Complexity of Bounded Subsumptions and Reductions	98
8	REDUCTION OF LEARNING EXAMPLES	104
8.1	Reduction of Examples for Mutually Non-Resolving Clausal Theories	104
8.2	Reduction of Examples for Mutually Resolving Clauses	106
8.3	Experimental Evaluation of Safe Reduction	107
8.3.1	Experiments with nFOIL	108
8.3.2	Experiments with Aleph	109
8.4	Conclusions	110
8.5	Proofs	110
IV	BIOINFORMATICS APPLICATIONS	114
9	PREDICTION OF DNA-BINDING PROPENSITY OF PROTEINS	115
9.1	Background and Related Work	115
9.2	Datasets	116
9.3	Method	116
9.4	Experiments, Results and Discussion	117
9.4.1	Evaluation of binding motif independence	119
9.5	Conclusions	121
10	PREDICTION OF ANTIMICROBIAL ACTIVITY OF PEPTIDES	122
10.1	Antimicrobial Peptides	122
10.2	Existing Approaches to Activity Prediction	123
10.3	Antimicrobial Activity Prediction	123

10.4	Data	124
10.5	Results	124
10.6	Conclusions	126
11	POLYNOMIAL FEATURES FOR PREDICTION OF DNA-BINDING PROPENSITY	127
11.1	Ball-Histogram Method	127
11.1.1	Ball histograms	127
11.1.2	Transformation-based learning	129
11.2	Extending Ball Histograms with Continuous Variables	129
11.2.1	Motivation	129
11.2.2	Multivariate Polynomial Aggregation Features	130
11.2.3	Ball Histograms with Continuous Variables	132
11.3	Experiments	132
11.3.1	Data	132
11.3.2	Results	133
11.4	Conclusions	135
11.5	Theoretical Details	135
V	CONCLUSIONS	136
12	CONCLUSIONS	137
VI	APPENDIX	138
A	TREELIKER - A SUITE OF FEATURE CONSTRUCTION ALGORITHMS	139
A.1	Implementation	139
A.1.1	Representation of Input Data	139
A.1.2	Types of Relational Features	140
A.1.3	Language Bias - Templates	140
A.1.4	TreeLiker GUI	142
A.1.5	TreeLiker from Command Line	143
A.1.6	Output	145
A.2	Availability and Requirements	145
B	A PROBABILISTIC FRAMEWORK FOR FEATURES	146
B.1	The Framework	146
B.2	Proofs of Propositions	148
	BIBLIOGRAPHY	150

INTRODUCTION

Relational machine learning is a subfield of machine learning which specializes on learning from complex structured data. Relational learning systems should ideally be able to deal with learning problems involving relations, numerical variables and uncertainty. A common problem of most relational learning systems is efficiency. One of the basic reasons for efficiency issues of relational learning systems is the generality and complexity of the problems they solve. In the most general settings of logic-based relational learning [23], which are detailed in Section 2.1, the learning problems are undecidable in general. Even decidable versions of relational learning problems usually have high complexity. Efficiency is thus a key problem that limits applicability of relational machine learning to real-world problems. In order to cope with the efficiency issues, many approaches have been devised but there is still a lot of space for further improvement. Efficiency of relational machine learning can be increased by several rather complementary strategies. For instance, it is possible to try to devise more efficient learning algorithms for the relational learning problems in their general form. Another possible strategy is to limit the problems to be solved. In this thesis, we are interested in this latter strategy.

This thesis deals with the problem of feature construction for relational machine learning. In the simplest form, relational features are simply first-order-logic formulas. They may be used for predictive classification as attributes in predictive models. They may also be used on their own for exploratory data analysis. This work revolves mainly around one basic question: how can we make construction of relational features more efficient when we assume that a meaningful set of features can be constructed from features from a limited class? The methods presented in this thesis, which were developed as a result of our attempts to offer at least a partial answer to this question, build mainly upon ideas from computational logic and constraint satisfaction [101]. They can be divided into two groups. Methods from the first group try to directly construct features only from the given class, for example from the class of all *treelike features*. These methods are described in Part ii. Methods from the second group do not necessarily construct features only from the given classes but they provide guarantees only in the form: if there is a feature from the given (possibly infinite) set then a possibly different feature at least as *good*¹ will be constructed. These methods bearing the common name *relational learning modulo hypothesis languages* are described in Part iii. The developed feature construction algorithms are relevant for real-world problems as we demonstrate in experiments with two important biological problems – the prediction of DNA-binding propensity of proteins and the prediction of antimicrobial activity of peptides. We show that our algorithms are able to automatically construct features for these problems which are better than published features hand-crafted by domain experts.

1.1 PROBLEM STATEMENT

The statement of the main problem tackled in this thesis is simple: *Make construction of relational features more efficient*. However, there are numerous ways to achieve this. One could try to make feature construction faster by developing faster algorithms for evaluation of constructed features, for example by speeding-up θ -subsumption (as we did in [68, 70]) which is a procedure often used as a *covering operator* for evaluation of relational features. One could also try to speed-up the construction of features by developing new search strategies which would be able to find good sets of features using stochastic methods like evolutionary algorithms. In this thesis, we are interested in neither of these two possibilities. Instead, we are interested in finding ways to make feature construction faster when we know or assume that a good set of features can be

¹ What we mean by *good* in this context will be explained in Part iii

constructed from features having some special form (e.g. from acyclic or treelike features). We are mostly interested in methods with provable theoretical guarantees.

1.2 MAIN CONTRIBUTIONS

The main contributions of this thesis are novel algorithms for construction of relational features and several other methods and ideas applicable in relational learning. The novel algorithms can be divided into two groups: algorithms based on a block-wise strategy exploiting *monotonicity of redundancy* and algorithms based on so-called *bounded least general generalization*. Algorithms from the first group are RelF [62, 71], HiFi [69] and Poly [63, 65]. These algorithms have been shown to outperform several state-of-the-art relational learning systems on many learning problems. An algorithm from the second group presented in this thesis is Bull. The main ideas of Bull were outlined in [66], however, its full description is presented for the first time in this thesis in Chapter 7.

Besides the main contributions in the form of new feature-construction algorithms, this thesis also studies the notion of multivariate polynomial aggregation. We show that polynomial aggregation features are useful in the relational context in conjunction with the algorithm Poly [63, 65] and also outside the relational context [64]. Another smaller contribution presented in this thesis is the introduction of the concept of *a safe reduction of a learning example* and the development of methods for its utilization as a preprocessing technique. We were able to speed up existing relational learning algorithms by preprocessing their input [61, 67].

We have also used some of the methods and ideas presented in this thesis as components of bioinformatics methods for solving the problems of the DNA-binding propensity prediction [118, 117, 64] and the antimicrobial activity prediction of peptides [119].

1.3 THESIS ORGANIZATION

This thesis is organized as follows. We describe the necessary theoretical minimum from logic-based relational learning and briefly present the most prominent existing relational learning systems in Chapter 2. Other existing relational learning systems not covered in this are described in more detail later in the ‘related-work’ sections of the subsequent chapters where they are compared to our new algorithms. Then we provide the necessary background on θ -subsumption and on constraint satisfaction problems and their relation to θ -subsumption in Chapter 3. Since there is no universal interpretation of what a relational feature is (and different interpretations may be useful for different problems), we study several types of relational features and discuss their properties in Chapter 4. The next two parts, Part ii and Part iii, in which our novel algorithms for construction of relational features are presented, constitute the core of the thesis. In Part ii, we present feature-construction algorithms based on a block-wise strategy for which we are able to prove monotonicity of redundancy which speeds up the feature construction process substantially. In this part, we describe two novel fast feature construction algorithms RelF and HiFi in Chapter 5 and another novel feature construction algorithm Poly intended for domains with significant amount of numerical information in Chapter 6. In Part iii, we describe a generic framework for relational learning based on weakening of standard notions of covering, reduction and generalization. In Chapter 7, we describe a novel bottom-up algorithm for construction of small sets of long features based on this generic framework. In Chapter 8, we show that the generic framework can be used for reduction of learning examples without affecting learnability. In Part iv, we present applications of our methods to important bioinformatics problems – to the problem of the prediction of DNA-binding propensity of proteins and to the problem of the prediction of antimicrobial activity of peptides. Finally, Part v concludes the thesis. In addition, there is a description of the open-source suite of the algorithms developed in this thesis called TreeLiker.

Part I

BACKGROUND

In this chapter, we introduce basic concepts of logic-based relational machine learning that directly relate to the methods presented in this thesis. We also review the most prominent existing relational learning approaches based on logic. Although there are also relational learning frameworks which are not based on the formalism of first-order logic, e.g. relational gaussian processes [18] or probabilistic relational models [37], the most well-known relational learning systems are based on first-order logic, e.g. Progol [86], Markov logic networks [28], Bayesian logic programs [55]. We should stress that not all existing methods and systems are covered in this chapter. Some are described in more detail in the respective ‘related work’ sections in the respective chapters where we can actually better explain their main properties by contrasting them with our methods.

2.1 LOGIC-BASED RELATIONAL MACHINE LEARNING

In this section, we briefly describe basic concepts of logic-based relational machine learning. We start by defining three learning settings: the classical *intensional inductive logic programming learning setting*, *learning from entailment* and *learning from interpretations setting*. [22].

Definition 1 (Covering under Intensional ILP). *Let H and B be clausal theories and e be a clause. Then we say that H covers e (w.r.t. B) if and only if $H \wedge B \models e$.*

Definition 2 (Covering under Learning from Entailment). *Let H be a clausal theory and e be a clause. Then we say that H covers e under entailment (denoted by $H \preceq_E e$) if and only if $H \models e$.*

Definition 3 (Covering under Learning from Interpretations). *Let H be a clausal theory and e a Herbrand interpretation. Then we say that H covers e under interpretations if and only if e is a model for H .*

The basic learning task is to find a clausal theory H which covers all positive examples and no negative example (under chosen learning setting). In this thesis we will use mainly the learning from interpretations setting but also the learning from entailment setting (e.g. in Chapters 7 and 8). We included the intensional inductive logic programming setting mainly because it is the most well-known setting.

Example 1 (Intensional Inductive Logic Programming). Let us have background knowledge

$$B = \{\text{flies}(\text{parrot}) \wedge \neg \text{flies}(\text{hippo})\}$$

and a set of positive examples (containing just one example in this case) $E^+ = \{\text{bird}(\text{parrot})\}$ and a set of negative example $E^- = \{\text{bird}(\text{hippo})\}$. Then a solution to the learning task under intensional inductive logic programming setting with the given sets of positive and negative examples is for example

$$H = \forall X : \text{flies}(X) \rightarrow \text{bird}(X).$$

We can easily verify that it holds $H \wedge B \models \text{bird}(\text{parrot})$ and $H \wedge B \not\models \text{bird}(\text{hippo})$. We can also verify that $H' = \forall X : \text{bird}(X)$ is not a valid solution because it covers a negative example ($H' \wedge B \models \text{bird}(\text{hippo})$).

Example 2 (Learning from Entailment). In the learning from entailment setting, there is no background knowledge. Let us assume that the sets of positive and negative examples, which are clauses, are

$$E^+ = \{\text{bird}(\text{parrot}) \leftarrow \text{flies}(\text{parrot})\}$$

and

$$E^- = \{\text{bird}(\text{hippo}) \leftarrow \neg \text{flies}(\text{hippo})\}.$$

The formula $H = \forall X : \text{flies}(X) \rightarrow \text{bird}(X)$ is a valid solution since

$$(\forall X : \text{flies}(X) \rightarrow \text{bird}(X)) \models (\text{bird}(\text{parrot}) \leftarrow \text{flies}(\text{parrot}))$$

and

$$(\forall X : \text{flies}(X) \rightarrow \text{bird}(X)) \not\models (\text{bird}(\text{hippo}) \leftarrow \neg \text{flies}(\text{hippo}))$$

(because there are models of H which are not models of $\text{bird}(\text{hippo}) \leftarrow \neg \text{flies}(\text{hippo})$).

Example 3 (Learning from Interpretations). We will now consider basically the same toy problem but this time we will use the learning from interpretations setting. In the learning from interpretations setting, there is no background knowledge. Let us assume that the sets of positive and negative examples, which are Herbrand interpretations, are

$$E^+ = \{\{\text{bird}(\text{parrot}), \text{flies}(\text{parrot})\}, \{\neg \text{bird}(\text{hippo}), \neg \text{flies}(\text{hippo})\}\}$$

and

$$E^- = \{\{\neg \text{bird}(\text{eagle}), \text{flies}(\text{eagle})\}\}.$$

Then, again, the formula $H = \forall X : \text{flies}(X) \rightarrow \text{bird}(X)$ is a valid solution since it holds that $\{\text{bird}(\text{parrot}), \text{flies}(\text{parrot})\}$ and $\{\neg \text{bird}(\text{hippo}), \neg \text{flies}(\text{hippo})\}$ are models for H and $\{\neg \text{bird}(\text{eagle}), \text{flies}(\text{eagle})\}$ is not a model for H . Similarly to the case of intensional inductive logic programming, $H' = \forall X : \text{bird}(X)$ is not a valid solution because then the second positive example would not be a model for this hypothesis H' .

We note that, in general, neither the intensional inductive logic programming setting [22] nor the learning from entailment are reducible to learning from interpretations. The main obstacle is that, unlike in the case of learning under intensional inductive logic programming, the examples must be fully specified in the learning under interpretations setting, i.e. the truth value of every ground atom must be known. On the other hand, in many real-life problems, learning from interpretations is a fully sufficient learning setting.

Of course, the ILP problem as described above is only an idealized problem which focuses only on the *search* aspect of learning. It does not take into account generalization performance of the learned theories. One approach to control generalization performance of learned theories which has been applied in practical implementations of logic-based relational machine learning [86] is to limit the set of allowed hypotheses either by imposing limits on maximum length of learned theories [86] or by considering only theories from a syntactically limited class - determinate theories, theories consisting of non-recursive definite clauses etc. Another approach is to apply methods developed within attribute-value statistical machine learning¹. For example systems presented in [73, 74] exploit the machinery of support vector machines [15]. Other systems, based on so-called *static propositionalization* can in principle exploit any method from attribute-value machine learning. These systems get a relational learning problem on their input and produce its attribute-value representation. For example, the systems from [25, 131] create an attribute-value table in which rows correspond to examples and columns correspond to first-order-logic formulas. There is a *true* in the entry (i, j) in this table if it holds that the j -th formula covers the i -th example. This table can be then fed into any classical attribute-value learning algorithm (e.g. SVM, decision tree etc.). *Propositionalization* is discussed in more detail in Section 2.2.

¹ We will use the term *attribute-value (statistical) machine learning* to denote any machine learning method whose input and output are (sets of) fixed-dimensional vectors.

2.2 LOGIC-BASED RELATIONAL LEARNING BASED ON PROPOSITIONALIZATION

The classical ILP settings outlined in Section 2.1, where an ILP system is given a set of positive examples E^+ , a set of negative examples E^- , possibly a background theory B and a language bias and its task is to find a theory H which covers all positive examples and no negative examples, are not very well suited for many practical problems. They do not cope well with uncertainty which may be caused by noise or by an inherent probabilistic nature of some learning problems. The formulations also do not take into account performance of the learned theories on unobserved data. Existing systems which more-or-less follow the classical settings such as Progol [86] have rather limited means to cope with these issues. For example, it is possible to set a tolerance for misclassification rate of the learned theories on training data and to set a maximum length of clauses in the learned theory in Progol. This can be used to control both noise and generalization - to some extent. Another possible way is to exploit techniques from attribute-value machine learning. *Propositionalization* is a framework, which enables one to exploit results from attribute value learning for classification learning. The basic idea behind *propositionalization* is to generate many first-order-logic formulas (called *features*) and to let these formulas act as attributes for attribute value learners. While the basic concept of propositionalization is quite straightforward, developing an efficient propositionalization system remains a hard task because several subproblems, which are encountered in propositionalization, are generally NP-hard. Propositionalization systems need to perform the following two basic problems: feature construction and coverage computation.

Let us now briefly review work on propositionalization - both *static* and *dynamic*. Several pattern mining algorithms have been proposed for exhaustive generation of large numbers of features in limited subsets of first order logic or for more or less similar tasks (static propositionalization). A well-known algorithm working in the logical setting is the frequent pattern miner WARMR [25], which greatly exploits monotonicity of frequency to prune uninteresting patterns. Another algorithm for pattern discovery is the RSD algorithm [131]. RSD does not prune patterns using the minimum frequency constraint as WARMR does, instead, RSD relies on its expressive user-definable syntactical bias, which is also somewhat similar to WARMR's warmode declarations. A common principle of RSD and WARMR is the level-wise approach to feature construction, which means that features are built by adding one logical atom at time.

A recent line of research, on so-called dynamic propositionalization, represented by algorithms nFOIL [74], kFOIL [73] and SAYU [21], tries to refrain from explicitly constructing the set of all *interesting* features (frequent, non-redundant etc.) by constructing only features that improve classification accuracy or some related scoring criterion when combined with a propositional learner such as Naive Bayes or SVM. Both nFOIL and kFOIL are based on FOIL's [99] search, although, in principle, any strategy for hypothesis search could be used instead of FOIL. A potential replacement for FOIL could be e.g. Progol [86] or even some procedure searching over features generated by a propositionalization algorithm. An advantage of systems like nFOIL and kFOIL is that they can produce relatively accurate models within reasonably short runtimes.

Frequent graph mining algorithms are also related to propositionalization. They are very well suited for molecular databases, however, they have limitations in other domains. For example, the currently fastest graph mining algorithm Gaston [93] is able to mine frequent molecular structures from large databases, but it cannot easily handle oriented graphs or even hypergraphs [134]. Another notable system geared towards discovery of patterns in molecular databases is MolFea [57], which restricts the patterns to linear molecular fragments.

Recently, there has been a growing interest in removing various forms of *redundancy* from frequent pattern sets. One form of *redundancy* was introduced in [77] where it was defined for propositional data and for constrained Horn clauses, which are equivalent to propositional representation in their expressive power. There are also other forms of *redundancy* considered in literature. For example, a system called Krimp was introduced in [56, 125] which searches for a representative set of features that would allow for a lossless compression of a given set of

examples, i.e. it relies on the minimum description length principle. Mining of closed frequent patterns is another approach that minimizes the number of discovered patterns by considering only representative candidates from certain equivalence classes.

2.3 STATISTICAL RELATIONAL LEARNING

The classical logic-based relational learning does not handle uncertainty very well. Statistical relational learning is a field that tries to combine relations (very often first-order-logic) with probability. This can be achieved, for example, by providing a function which would assign probabilities to Herbrand interpretations. Markov logic networks [28] and Bayesian logic programs [55] are instances of this approach.

One of the most important and well-known statistical relational learning systems is Markov logic [28]. A Markov logic network is given by a set of constants², a set of predicate symbols and a set of first-order-logic clauses (features) and their weights. The probability of a Herbrand interpretation x is then given by the formula

$$P(X = x) = \frac{1}{Z} \exp \left(\sum_i w_i n_{F_i}(x) \right)$$

where $n_{F_i}(x)$ is the number of true groundings of F_i in x and w_i is the weight of the feature F_i , Z is a normalization constant known also as *partition function* and can be expressed as $Z = \sum_{x \in \mathcal{X}} \exp(\sum_i w_i n_{F_i}(x))$ (here, \mathcal{X} is the set of all the possible worlds - interpretations, constructed using the declared predicate symbols and constants). A Markov logic network can be also viewed as a recipe for construction of ordinary Markov networks. A Markov logic network corresponds to an ordinary Markov network where the set of nodes (random variables) corresponds to the set of all possible groundings of all predicate symbols (using the constants from the given Markov logic network). The random variables (nodes) are two-valued - having one of the two possible values *true* or *false* which denote whether the given ground fact is true or false in the Herbrand interpretation. This correspondence with ordinary Markov networks makes it possible to use inference methods, e.g. Gibbs sampling, developed for the ordinary Markov networks. It also provides ways to find a set of ground atoms that are needed to answer a (conditional) query. Markov logic networks can be learned from data, typically using optimization of weighted pseudo-likelihood to select weight for features and some form of structure learning for the first-order-logic formulas. There are other statistical relational learning systems, for example: Bayesian logic programs [55], Probabilistic relational models [37], relational dependency networks [91]. Most of these systems can, however, be emulated within the framework of Markov logic.

There is also an extension of Markov logic networks called Hybrid Markov logic networks [132] that enables working with continuous variables within Markov logic by extending their syntax and adding new types of features. However, there is no structure learning algorithm (i.e. an algorithm for learning the logical formulas) for Hybrid Markov logic which we conjecture to be a consequence of the very high complexity of this framework. Indeed, experiments reported in literature for Hybrid Markov logic were performed on rather small problems [132]. There is also no exact inference method for Hybrid Markov logic. Hybrid Markov logic is not the only framework capable to model probability distributions over relational structures containing continuous numerical data. Bayesian logic programs [55] allow rather seamless incorporation of continuous variables as well. However, again to our best knowledge, there has not been any specific proposal for a learning strategy that would efficiently deal with continuous random variables. Recently, continuous distributions have also been incorporated into the Problog system [42]. Finally, in [18] relational Gaussian processes were introduced which were later extended

² Markov logic networks define probability over finite interpretations. However, there has also been a theoretical work [27] which introduced a generalization of Markov logic to infinite domains.

to the multi-relational setting in [136]. Relational Gaussian processes are able to work with continuous random variables. Relational Gaussian processes are non-parametric which also means that no model selection, i.e. structure search, is necessary.

2.4 FUNCTION-FREENESS ASSUMPTION

In this thesis, we assume that clauses do not contain any function symbols other than constants. This is not too strict a restriction as it is possible to use *flattening* to transform a learning problem involving clauses with function symbols to a problem without function symbols [92]. At some places in this thesis, we state explicitly that we consider only function-free clauses when there is a risk of confusion but we use it implicitly at other places unless explicitly stated otherwise.

The undecidable entailment relation \models is often replaced in relational learning systems by its decidable approximation called θ -subsumption which was introduced by Plotkin [97]. Nevertheless θ -subsumption is also not easy to decide as it is an NP-complete problem. In this chapter we describe how θ -subsumption can be solved relatively efficiently using the machinery of constraint satisfaction. We also review basic tractability results for constraint satisfaction problems with bounded treewidth and explain that these results also apply for the θ -subsumption problem. Nothing in this chapter is new. A tight relationship between θ -subsumption and constraint satisfaction has been known for a long time [31, 82].

3.1 θ -SUBSUMPTION AND THE SUBSUMPTION THEOREM

Let us first state some notational conventions. A first-order-logic clause is a universally quantified disjunction of first-order-logic literals. For convenience, we do not write the universal quantifiers explicitly. We treat clauses as disjunctions of literals and as sets of literals interchangeably. We will sometimes use a slightly abused notation $a(x, y) \subseteq a(w, x) \vee a(x, y)$ to denote that a set of literals of one clause is a subset of literals of another clause. The set of variables in a clause A is written as $\text{vars}(A)$ and the set of all terms by $\text{terms}(A)$. Terms can be variables or constants. A substitution θ is a mapping from variables of a clause A to terms of a clause B . A set of clauses is said to be *standardized-apart* if no two clauses in the set share a variable.

If A and B are clauses, then we say that the clause A θ -subsumes the clause B , if and only if there is a substitution θ such that $A\theta \subseteq B$. If $A \preceq_{\theta} B$ and $B \preceq_{\theta} A$, we call A and B θ -equivalent (written $A \approx_{\theta} B$). The notion of θ -subsumption was introduced by Plotkin [97] as an incomplete approximation of implication. Let A and B be clauses. If $A \preceq_{\theta} B$ then $A \models B$ but the other direction of the implication does not hold in general. However, it does hold for non-self-resolving function-free clauses. Another way to look at $A \preceq_{\theta} B$ is to view it as homomorphism between relational structures. Informally, $A \preceq_{\theta} B$ holds if and only if there is a homomorphism $A \rightarrow B$ where A and B are treated as relational structures.

If A is a clause and there is another clause R such that $A \approx_{\theta} R$ and $|R| < |A|$ then A is said to be θ -reducible. A minimal such R is called θ -reduction of A . For example, the clause

$$A = \text{east}(T) \leftarrow \text{hasCar}(T, C) \wedge \text{hasLoad}(C, L1) \wedge \text{hasLoad}(C, L2) \wedge \text{box}(L2)$$

is θ -reducible because $A \approx_{\theta} B$ where

$$B = \text{east}(T) \leftarrow \text{hasCar}(T, C) \wedge \text{hasLoad}(C, L) \wedge \text{box}(L).$$

In this case, B is also a θ -reduction of A . The concept of θ -reduction of clauses is equivalent to the concept of *cores* [4] of relational structures. *Core* of a relational structure A is a minimal relational structure homomorphically equivalent to A .

A precise relationship between logical entailment, θ -subsumption and first-order resolution is given by the so-called *subsumption theorem* [92]. Before stating this theorem, we need to briefly describe first-order resolution. Our description follows the description from [92]. We start the description by introducing some auxiliary concepts which are used in resolution.

Definition 4 (Most general unifier). *Let $L = \{l_1, l_2, \dots, l_k\}$ be a set of literals. A substitution ϑ is called a unifier of L if and only if $l_1\vartheta = l_2\vartheta = \dots = l_k\vartheta$. A substitution θ is a most general unifier of L if and only if for any unifier θ' of L there is a unifier θ'' such that $l_1\theta' = l_1\theta\theta''$.*

Importantly, one can always select a most general unifier θ which maps variables of all literals from L to variables which are not present in the literals in L . Such most general unifiers will be of use in Chapter 8.

Definition 5 (Binary resolvent). *Let $A = l_1 \vee l_2 \vee \dots \vee l_m$ and $B = m_1 \vee m_2 \vee \dots \vee m_n$ be two standardized-apart clauses. Let θ be a most general unifier of the literals $l_i, \neg m_j$ such that $\text{vars}(l_i\theta) \cap \text{vars}(A) = \text{vars}(l_i\theta) \cap \text{vars}(B) = \emptyset$. Next, let*

$$C = (l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_m \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)\theta$$

Then C is called a binary resolvent of A and B .

The next definition introduces the notion of a *resolvent* which is more general than the notion of a *binary resolvent*.

Definition 6 (Resolvent). *Let $A = l_1 \vee l_2 \vee \dots \vee l_m$ be a clause and let $L = \{l_1, l_2, \dots, l_k\} \subseteq A$ be a set of unifiable literals. If θ is a most general unifier of L then the clause obtained as $A\theta \setminus \{l_1, \dots, l_k\}\theta$ is called a factor of A . If B and C are clauses then a resolvent of B and C is a binary resolvent of a factor of B and a factor of C where the literals resolved upon are the literals unified by the respective factors.*

Now, we can define what is meant by deriving a clause by resolution or by SLD-resolution.

Definition 7 (Derivation by resolution). *Let S be a set of clauses and A be a clause. A derivation of the clause A from S by resolution is a finite sequence of clauses R_1, R_2, \dots, R_k where $R_k = A$ such that each R_i is either in S or a resolvent of two clauses in the subsequence R_1, R_2, \dots, R_{i-1} .*

Definition 8 (Derivation by SLD-resolution). *Let S be a set of Horn clauses and A be a Horn clause. A derivation of the clause A from S by SLD-resolution is a finite sequence of Horn clauses R_1, R_2, \dots, R_k where $R_k = A$ such that $R_0 \in S$ and each R_i is a binary resolvent of R_{i-1} and a definite clause (i.e. a Horn clause having exactly one positive literal) $C_i \in S$ where the literals resolved upon are the literal in the head of C_i and a selected negative literal from R_{i-1} .*

Finally, we can state the subsumption theorem for resolution and for SLD-resolution where the latter ‘works’ only for Horn clauses.

Proposition 1 (Subsumption theorem [92]). *Let S be a set of clauses and A be a clause which is not a tautology. Then $S \models A$ if and only if there is a clause B derivable by resolution from S such that $B \preceq_\theta A$.*

Proposition 2 (Subsumption theorem for SLD-resolution [92]). *Let S be a set of Horn clauses and A be a Horn clause which is not a tautology. Then $S \models A$ if and only if there is a clause B derivable by SLD resolution from S such that $B \preceq_\theta A$.*

Subsumption theorems explain the connections between θ -subsumption, resolution and logical entailment. We will need them in Chapter 8.

3.2 θ -SUBSUMPTION AS A CONSTRAINT SATISFACTION PROBLEM

Constraint satisfaction [24] with finite domains represents a class of problems closely related to the θ -subsumption problems and to relational-structure homomorphisms. In fact, as shown by Feder and Vardi [31], these problems are almost identical although the terminology differs. This equivalence of CSP and θ -subsumption has been exploited by Maloberti and Sebag [82] who used off-the-shelf CSP algorithms to develop a fast θ -subsumption algorithm.

Definition 9 (Constraint Satisfaction Problem). *A constraint satisfaction problem is a triple $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, where \mathcal{V} is a set of variables, $\mathcal{D} = \{D_1, \dots, D_{|\mathcal{V}|}\}$ is a set of domains of values (for each variable $v \in \mathcal{V}$), and $\mathcal{C} = \{C_1, \dots, C_{|\mathcal{C}|}\}$ is a set of constraints. Every constraint is a pair (s, R) , where s (scope) is an n -tuple of variables and R is an n -ary relation. An evaluation of variables θ satisfies a constraint $C_i = (s_i, R_i)$ if $s_i\theta \in R_i$. A solution is an evaluation that maps all variables to elements in their domains and satisfies all constraints.*

The CSP representation of the problem of deciding $A \preceq_{\theta} B$ has the following form. There is one CSP variable X_v for every variable $v \in \text{vars}(A)$. The domain of each of these CSP variables contains all terms from $\text{terms}(B)$. The set of constraints contains one k -ary constraint $C_l = (s_l, R_l)$ for each literal $l = \text{pred}_l(t_1, \dots, t_k) \in A$. We denote by $I_{\text{var}} = (i_1, \dots, i_m) \subseteq (1, \dots, k)$ the indexes of variables in arguments of l (the other arguments might contain constants). The scope s_l of the constraint C_l is $(X_{t_{i_1}}, \dots, X_{t_{i_m}})$ (i.e. the scope contains all CSP variables corresponding to variables in the arguments of literal l). The relation R_l of the constraint C_l is then constructed in three steps.

1. A set L_l is created which contains all literals $l' \in B$ such that $l \preceq_{\theta} l'$ (note that checking θ -subsumption of two literals is a trivial linear-time operation).
2. Then a relation R_l^* is constructed for every literal $l \in A$ from the arguments of literals in the respective set L_l . The relation R_l^* contains a tuple of terms (t'_1, \dots, t'_k) if and only if there is a literal $l' \in L_l$ with arguments (t'_1, \dots, t'_k) .
3. Finally, the relation R_l of the constraint C_l is the projection of R_l^* on indexes I_{var} (only the elements of tuples of terms which correspond to variables in l are retained).

If we do not specify otherwise, when we refer to *CSP encoding* of θ -subsumption problems in the remainder of this chapter, we will implicitly mean this encoding. Next, we exemplify the transformation process.

Example 4 (Converting θ -subsumption to CSP). Let us have clauses A and B as follows

$$A = \text{hasCar}(C) \vee \text{hasLoad}(C, L) \vee \text{shape}(L, \text{box})$$

$$B = \text{hasCar}(c) \vee \text{hasLoad}(c, l_1) \vee \text{hasLoad}(c, l_2) \vee \text{shape}(l_2, \text{box}).$$

We now show how we can convert the problem of deciding $A \preceq_{\theta} B$ to a CSP problem. Let $\mathcal{V} = \{C, L\}$ be a set of CSP-variables and let $\mathcal{D} = \{D_C, D_L\}$ be a set of domains of variables from \mathcal{V} such that $D_C = D_L = \{c, l_1, l_2\}$. Further, let $\mathcal{C} = \{C_{\text{hasCar}(C)}, C_{\text{hasLoad}(C,L)}, C_{\text{shape}(L,\text{box})}\}$ be a set of constraints with scopes (C) , (C, L) and (L) and with relations $\{(c)\}$, $\{(c, l_1), (c, l_2)\}$ and $\{(l_2)\}$, respectively. Then the constraint satisfaction problem given by \mathcal{V} , \mathcal{D} and \mathcal{C} represents the problem of deciding $A \preceq_{\theta} B$ as it admits a solution if and only if $A \preceq_{\theta} B$ holds.

3.3 TREE DECOMPOSITIONS AND TREEWIDTH

The Gaifman (or primal) graph of a clause A is the graph with one vertex for each variable $v \in \text{vars}(A)$ and an edge for every pair of variables $u, v \in \text{vars}(A)$, $u \neq v$ such that u and v appear in a literal $l \in A$. Similarly, we define Gaifman graphs for CSPs. The Gaifman graph of a CSP problem $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ is the graph with one vertex for each variable $v \in \mathcal{V}$ and an edge for every pair of variables which appear in a scope of some constraint $c \in \mathcal{C}$. Gaifman graphs can be used to define treewidth of clauses or CSPs.

Definition 10 (Tree decomposition of a Graph, Treewidth). *A tree decomposition of a graph $G = (V, E)$ is a tree T with nodes labelled by sets of vertices such that*

- For every vertex $v \in V$, there is a node of T with a label containing v .
- For every edge $(v, w) \in E$, there is a node of T with a label containing v, w .
- For every $v \in V$, the set of nodes of T with labels containing v is a connected subgraph of T .

The width of a tree decomposition T is the maximum cardinality of a label in T minus 1. The treewidth of a graph G is the smallest number k such that G has a tree decomposition of width k . The treewidth of a clause is equal to the treewidth of its Gaifman graph. Analogically, the treewidth of a CSP is equal to the treewidth of its Gaifman graph.


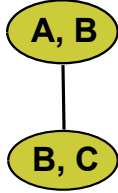
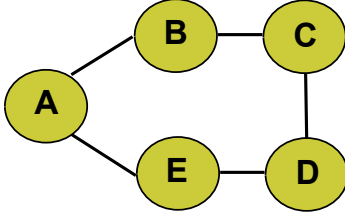
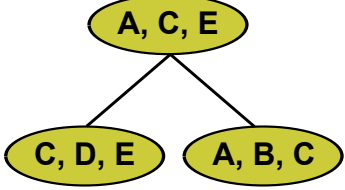
Clause	Gaifman graph	Tree decomposition
$\leftarrow \text{atm}(A, h) \wedge$ $\text{bond}(A, B, 1) \wedge \text{atm}(B, c) \wedge$ $\text{bond}(B, C, 2) \wedge \text{atm}(C, o)$		
$\leftarrow \text{bond}(A, B, 1) \wedge$ $\text{bond}(B, C, 1) \wedge \text{bond}(C, D, 1) \wedge$ $\text{bond}(D, E, 1) \wedge \text{bond}(E, A, 1)$		

Table 1: An illustration of Gaifman graphs and tree decompositions of clauses.

An illustration of Gaifman graphs of two exemplar clauses and their tree decompositions is shown in Table 1. Note that tree decompositions are not unique. That is why treewidth is defined as the *maximum* cardinality of a label minus 1.

For instance, all trees have treewidth 1, cycles have treewidth 2, rectangular $n \times n$ grid-graphs have treewidth n . Treewidth is usually used to isolate tractable sub-classes of NP-hard problems. A general result about polynomial-time solubility of some problems on graphs of bounded treewidth is Courcelle's proposition [19] which essentially states that every problem definable in Monadic Second-Order logic can be solved in linear time on graphs of bounded treewidth (though, the run-time is exponential in the treewidth parameter of the graphs). Constraint satisfaction problems with treewidth bounded by k can be solved in polynomial time by the k -consistency algorithm¹.

Sometimes, it will be convenient to use the following definition of *tree decomposition of a clause* which does not directly refer to the concept of a Gaifman graph. The *treewidth* of a clause given by this definition and the *treewidth* given by Definition 10 are equivalent.

Definition 11 (Tree decomposition of a Clause). *A tree decomposition of a clause A is a tree T with nodes labelled by sets of first-order-logic variables such that*

- For every variable $V \in \text{vars}(A)$, there is a node of T with a label containing V .
- For every pair of variables $U, V \in \text{vars}(A)$ such that $U \neq V$ and U, V are both contained in a literal $\iota \in A$, there is a node of T with label containing U, V .
- For every $V \in \text{vars}(A)$, the set of nodes of T with labels containing V is a connected subgraph of T .

The width of a tree decomposition T is the maximum cardinality of a label in T minus 1. The treewidth of a clause A is the smallest number k such that A has a tree decomposition of width k .

Another class of CSPs for which efficient algorithms exist is represented by so-called acyclic CSPs. In order to define the notion of *acyclic CSPs*, we need to introduce the concept of *primal hypergraph* of a CSP which can be seen as a hypergraph parallel to Gaifman graph. The primal hypergraph of a CSP problem $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ is the hypergraph with one vertex for every variable $v \in \mathcal{V}$ and one hyperedge for every constraint $C = (s, R) \in \mathcal{C}$ where the hyperedge for a

¹ In this chapter we follow the conventions of [4]. In other works, what we call k -consistency is known as *strong* $k+1$ -consistency [101].

constraint C contains exactly the vertices corresponding to the variables in its scope s . A CSP problem is said to be acyclic if its primal hypergraph is acyclic. The class of acyclic CSPs differs from the class of CSPs with treewidth 1 when the arity of constraints in them is greater than two. For precise characterization of acyclic hypergraphs, we follow the established definition from [29]. We also translate the notions into the context of clauses because, similarly as in the case of bounded-treewidth clauses, if a clause is acyclic then also its CSP encoding is acyclic.

Definition 12 (Acyclic clause, acyclic hypergraph). *A clause (hypergraph) C is treelike if the iteration of the following rules on C produces the empty conjunction (hypergraph):*

1. *Remove a hyperedge contained in another hyperedge (a literal which shares all its variables with another literal).*
2. *Remove a vertex (variable) which is contained in at most one hyperedge (literal).*

Analogically to the case of bounded treewidth of clauses, if a clause is acyclic then its CSP encoding is also acyclic. Acyclic CSPs with extensionally represented relations of constraints can be solved in time polynomial in the size of the problem by the so-called *generalized arc-consistency algorithm* [6].

In this chapter, we describe four types of relational features and study their properties. Two of the described types of features are well known, namely *Boolean features* often used in classical inductive logic programming and *counting features* used in Markov logic [28]. The remaining two types of features are to our best knowledge less common¹. These are *polynomial features* and *generalized multivariate aggregation features*. What we would like to make clear in this chapter is that there is no single universal concept of a relational feature and that different types of relational features can be useful for different applications.

Two of the four types of features described in this chapter, *polynomial features* and *generalized multivariate aggregation features*, aim mainly at learning in so-called *hybrid domains*, i.e. in relational domains which contain substantial part of information in the numeric form. Learning in hybrid relational domains is an important problem with applications in areas as different as bioinformatics or finance. So far there have not been many relational learning systems introduced in literature that would be able to model multi-relational domains with numerical data efficiently. One of the frameworks able to work in such domains is hybrid Markov logic [132]. Another type of systems suitable for hybrid domains is represented by systems based on relational aggregation (e.g. [58]). *Polynomial features* presented in this chapter, are related both to hybrid Markov logic and to aggregation-based systems as we shall discuss in the respective sections.

This chapter is organized as follows. We start by describing the minimum necessary notation in Section 4.1. Then we describe a simple probabilistic framework for learning in relational domains possibly containing numerical data in Section 4.2 which constitutes theoretical foundations for the subsequent sections. Descriptions of Boolean, counting, polynomial and generalized multivariate aggregation features are given in Sections 4.3, 4.4, 4.5 and 4.6. The chapter is concluded in Section 4.8. We do not present algorithms for construction of features or for their evaluation in this chapter. These algorithms are presented in the subsequent chapters. This chapter mainly serves to introduce the different types of features, to point out their strengths and weaknesses and to provide basis for the developments in the subsequent chapters.

4.1 NOTATION

Here, we present the minimum notation needed in this chapter. Let $n \in \mathbb{N}$. If $\vec{v} \in \mathbb{R}^n$ then v_i ($1 \leq i \leq n$) denotes the i -th component of \vec{v} . If $I \subseteq [1; n]$ then $\vec{v}_I = (v_{i_1}, v_{i_2}, \dots, v_{i_{|I|}})$ where $i_j \in I$ ($1 \leq j \leq |I|$). To describe training examples as well as learned models, we use a conventional first-order logic language \mathcal{L} whose alphabet contains a distinguished set of constants $\{r_1, r_2, \dots, r_n\}$ and variables $\{R_1, R_2, \dots, R_m\}$ ($n, m \in \mathbb{N}$). An r -substitution ϑ is any substitution as long as it maps variables (other than) R_i only to terms (other than) r_j . For the largest k such that $\{R_1/r_{i_1}, R_2/r_{i_2}, \dots, R_k/r_{i_k}\} \subseteq \vartheta$ we denote $I(\vartheta) = (i_1, i_2, \dots, i_k)$. A (Herbrand) *interpretation* is a set of ground atoms of \mathcal{L} . $I(H)$ ($I(F)$) denotes the naturally ordered set of indexes of all constants r_i found in an interpretation H (\mathcal{L} -formula F).

¹ For instance, polynomial have never been used in the context in which we use them, i.e. in propositionalization. On the other hand, somewhat similar ideas have already been used in statistical relational learning, namely in hybrid Markov logic [132].

4.2 A SIMPLE PROBABILISTIC FRAMEWORK

Models in our formalism² are split into two parts: discrete and continuous. Our training examples have both *structure* and *real parameters*. An example may e.g. describe a measurement of the expression of several genes; here the structure would describe functional relations between the genes and the parameters would describe their measured expressions. The structure is described by an interpretation, in which the constants r_i represent uninstantiated real parameters. The parameter values are determined by a real vector. Formally, an example is a pair $(H, \vec{\theta})$ where H is an interpretation, $\vec{\theta} \in \Omega_H$, and $\Omega_H \subseteq \mathbb{R}^{|\mathcal{I}(H)|}$.

Examples are assumed to be sampled from a distribution with the probability function

$$P(H, \Omega_H) = \int_{\Omega_H} f_H(\vec{\theta}|H) P(H) d\vec{\theta} \quad (1)$$

Here, $P(H)$ is a discrete probability distribution on the countable set of finite Herbrand interpretations of \mathcal{L} . If \mathcal{L} has functions other than constants, we assume that $P(H)$ is non-zero only for finite H . $f_H(\vec{\theta}|H)$ are the conditional densities of the parameter values. The advantage of this definition is that it cleanly splits the possible-world probability into the discrete part $P(H)$ which can be modeled by state-of-the-art approaches such as Markov Logic Networks (MLNs) [28], and the continuous conditional densities $f_H(\vec{\theta}|H)$. Intuitively, we can imagine this probabilistic model as defining a procedure for sampling learning examples as follows. First, the structure H is drawn randomly according to $P(H)$, then the right probability density function f_H corresponding to the structure H is found in a given possibly infinite lookup table and a vector $\vec{\theta}$ of numerical parameter is drawn randomly according to f_H .

A seemingly more elegant representation of learning examples would be obtained by substituting the numerical parameters from $\vec{\theta}$ for the respective distinguished constants r_i in H . Then, a learning example would be simply represented as $H\vec{\theta}$ (here $\vec{\theta}$ is meant as a substitution), as opposed to how they are represented in our framework – as pairs $(H, \vec{\theta})$. However, this would not work without serious problems. For instance, if we wanted to draw a random example using this alternative formalism, we would draw its structure H randomly from the distribution $P(H)$, then we would select a random sample of values of numeric variables according to $f_H(\vec{\theta}|H)$ and finally we would substitute the values from $\vec{\theta}$ for the distinguished constants in H . The problem is that the number of numeric parameters might unexpectedly decrease in some singular cases as the next example illustrates.

Example 5. Let $P(H)$ be a distribution such that $P(H_1) = 1$ and $P(H_i) = 0$ for any $H_i \neq H_1$ where

$$H_1 = \{a(r_1), a(r_2)\}.$$

Let $f_H(\vec{\theta}|H)$ be a continuous distribution, for instance, a bivariate normal distribution with zero mean and diagonal covariance matrix. Now, let us assume that we want to draw an example from the distribution given by $P(H)$ and $f_H(\vec{\theta}|H)$. We get the structure $H = H_1$ and draw a random vector $\vec{\theta}$ from the distribution given by $f_H(\vec{\theta}|H)$. Let us assume that $\vec{\theta} = (1, 1)$. We substitute for r_1 and r_2 and obtain an example $e = \{a(1)\}$ (because Herbrand interpretations are sets and not multi-sets). As we can see, e contains only one number, instead of two which we would expect according to the dimension of the continuous part of the distribution. This would be a problem for analyses presented in the subsequent sections (we would have to treat many singular cases individually). Fortunately, such problems do not occur when we use the framework which separates the structure H and the real parameters $\vec{\theta}$.

We are not primarily interested in learning the distributions. Instead, we are more interested in mining various types of local patterns (features) which might be useful for capturing characteristic properties of these distributions. Such local patterns can be used either for discriminative classification or as building blocks of global models.

² A rigorous treatment of the probabilistic formalism introduced here is provided in Appendix.

We need features to be able to somehow extract numerical values from learning examples. *Sample sets* are constructs which formalize *extraction* of numerical parameters from learning examples.

Definition 13 (Sample set). *Given an example $e = (H, \vec{\theta})$ and a feature F , the sample set of F and e is the multi-set $\mathcal{S}(F, e) = \{\vec{\theta}_{I(\vartheta)} | H \models F\vartheta\}$ where ϑ are r -substitutions grounding all free variables³ in φ , and $H \models F\vartheta$ denotes that $F\vartheta$ is true under H .*

Note that when a feature F contains no distinguished variables then its sample set w.r.t. an example $e = (H, \vec{\theta})$ contains k copies of the empty vector where k is the number of r -substitutions ϑ such that $H \models F\vartheta$.

Example 6. Let us have a feature

$$F = g(G_1, R_1) \wedge g(G_2, R_2) \wedge \text{expr}(G_1, G_2)$$

and an example $e = (H, \vec{\theta})$ describing a group of genes and the functional relations among them where

$$\begin{aligned} H &= \{g(g_1, r_1), g(g_2, r_2), g(g_3, r_3), g(g_4, r_4), \text{expr}(g_1, g_2), \\ &\quad \text{expr}(g_2, g_3), \text{expr}(g_3, g_4)\} \\ \vec{\theta} &= (1, 1, 0, 1) \end{aligned}$$

The sample set of the feature F w.r.t. the example e is

$$\mathcal{S}(F, e) = \{(1, 1), (1, 1), (0, 1)\}.$$

Sample sets can be used to compute relational counterparts of *mean*, *variance*, *covariance* etc. Importantly, as we shall see later, there is a well-defined meaning of these statistics when treating them within the probabilistic framework described in this section.

4.3 BOOLEAN FEATURES

The term *Boolean feature* does not relate so much to structure of features but to how we interpret them. Let F be a feature and let $e = (H, \vec{\theta})$ be an example. If F contains no distinguished variables and no distinguished constants then we say that F covers e if and only if $H \models F$. If F contains distinguished variables and $\vec{\theta}$ is a vector then we say that $(F, \vec{\theta})$ covers e if and only if $\vec{\theta} \in \mathcal{S}(F, e)$. Similarly, if Ω is a set of vectors then we say that (F, Ω) covers e if and only if $\Omega \cap \mathcal{S}(F, e) \neq \emptyset$. Notice that covering of an example by a feature with no distinguished variables or constants is the covering relation from the *learning from interpretations* setting of inductive logic programming [22]. The other two covering relations are exemplified next.

Example 7. Let us have a feature

$$F = \text{size}(X, R_1) \wedge \text{edge}(X, Y) \wedge \text{size}(Y, R_2),$$

and an example $e = (H, \vec{\theta})$

$$\begin{aligned} H &= \{\text{size}(a, r_1), \text{size}(b, r_2), \text{size}(c, r_3), \text{edge}(a, b), \text{edge}(b, c)\} \\ \vec{\theta} &= (1, 0, 3) \end{aligned}$$

Let $\vec{\theta}_1 = (1, 0)$ and $\vec{\theta}_2 = (1, 1)$ be vectors. Then $(F, \vec{\theta}_1)$ covers the example e because $\vec{\theta}_1 \in \mathcal{S}(F, e)$, but $(F, \vec{\theta}_2)$ does not cover e . Similarly, let $\Omega = \{(1, 1), (1, 0)\}$ be a set of vectors. Then (F, Ω) covers the example e because $\Omega \cap \mathcal{S}(F, e) = \{(1, 0)\} \neq \emptyset$.

³ Note that an interpretation H does not assign domain elements to variables in \mathcal{L} . The truth value of a *closed* formula (i.e., one where all variables are quantified) under H does not depend on variable assignment. For a general formula though, it does depend on the assignment to its free (unquantified) variables.

In more classical relational learning settings (e.g. in learning from interpretations), the example e would be described simply as an interpretation

$$e = \{\text{size}(a, 1), \text{size}(b, 0), \text{size}(c, 3), \text{edge}(a, b), \text{edge}(b, c)\}$$

and what we represent by the pair $(F, \vec{\theta}_1)$ would normally be represented as

$$F = \exists X, Y : \text{size}(X, 1) \wedge \text{edge}(X, Y) \wedge \text{size}(Y, 0).$$

It is easy to check that $e \models F$ (this corresponds to the covering relation under learning from interpretations). Similarly, (F, Ω) could be represented as

$$F = \exists X, Y : \text{size}(X, R_1) \wedge \text{edge}(X, Y) \wedge \text{size}(Y, R_2) \wedge R_1 = 1 \wedge (R_2 = 1 \vee R_2 = 0).$$

Recall that the main reason why we use the formalism in which structure and real parameters are kept separately is that it is simpler to work with probabilistic distributions involving both structure and real parameters when they are separated.

As we are mostly interested in using features as local patterns, we need to explain how they relate to the introduced probabilistic framework. We start by explaining what we will mean by *probability of a Boolean feature*. If a feature F contains no distinguished variables then its *probability* is simply the probability that a structure H is drawn from the given distribution $P(H)$ such that $H \models F$. Thus, the *probability* of a feature F without distinguished variables and distinguished constants can be computed as

$$P_F(F) = \sum_{H \in \{H' \mid H' \models F\}} P(H).$$

The situation is more complicated when F contains distinguished variables. Then we need to define probability of a pair (F, Ω) where F is a feature and Ω is a measurable subset of \mathbb{R}^n with n being equal to the number of distinguished variables in F . The *probability* of such pair is defined as being equal to the probability that an example $e = (H, \vec{\theta})$ is sampled from the given distribution such that $\mathcal{S}(F, e) \cap \Omega \neq \emptyset$. Formally,

$$P_F(F, \Omega) = \sum_H P(H) \cdot \int_{\{\vec{\theta} \mid \mathcal{S}(F, (H, \vec{\theta})) \cap \Omega \neq \emptyset\}} f_H(\vec{\theta} \mid H) d\vec{\theta}$$

What holds universally for Boolean features is the following simple property: If F and G are features and $F \preceq_{\theta} G$ then $P_F(F, \Omega) \geq P_F(G, \Omega)$ for any Ω .

The Boolean interpretation of features is useful for some types of problems but not so useful for other types as the next example demonstrates.

Example 8. Let us have a distribution on learning examples given by $P(H, \Omega) = \int_{\Omega} f_H(\vec{\theta} \mid H) P(H) d\vec{\theta}$. Let $P(H)$ be given as follows:

$$P(H) = \begin{cases} 0.5 \dots & \text{if } H = \{\text{num}(r_1)\} \\ 0.5 \dots & \text{if } H = \{\text{num}(r_1), \text{num}(r_2)\} \\ 0 \dots & \text{otherwise} \end{cases}$$

and let

$$\begin{aligned} f_H(\vec{\theta} \mid \{\text{num}(r_1)\}) &= 1 \text{ for } \vec{\theta} \in [0; 1] \\ f_H(\vec{\theta} \mid \{\text{num}(r_1), \text{num}(r_2)\}) &= 1 \text{ for } \vec{\theta} \in [0; 1] \times [0; 1]. \end{aligned}$$

Next, let us suppose that we have a feature $F = \text{num}(R_1)$ and a set $\Omega = [0, 0.5]$. We compute the probability of the pair (F, Ω) conditioned on the structure of the example and get the following results

$$\begin{aligned} P_F(F, \Omega \mid H = \{\text{num}(r_1)\}) &= 0.5 \\ P_F(F, \Omega \mid H = \{\text{num}(r_1), \text{num}(r_2)\}) &= 0.75. \end{aligned}$$

We may notice in the above example that the probability of the pair (F, Ω) increased with the number of true groundings (w.r.t. the given example) of the feature F . In general, this would also depend on the conditional probability density function associated to the particular learning-example structure H . However, what is important in this exemplified case is that the probability grew even though the individual continuous random variables were identically and independently distributed.

Whether the properties of Boolean features illustrated above are desirable depends on the particular application at hand. Boolean features may be useful for those applications where the property of interest depends on the presence of a certain substructure with the right values of the corresponding numerical parameters – for instance, if a property of a molecule, e.g. its toxicity, depends on the presence or absence of certain configuration of atoms. It may be less useful for applications where, for instance, the property of interest depends on the overall *behaviour* of numerical parameters.

4.4 COUNTING FEATURES

Counting interpretation of features gives us another way to look at features. When dealing with counting features, there is no longer a Boolean *covers* relation. Instead, there is a function which assigns a number to a pair (F, Ω) and an example e^4 . This number is called *count* of (F, Ω) w.r.t. the example e and is defined as follows. If F is a feature, Ω is a subset of \mathbb{R}^n where n is equal to the number of distinguished variables of F and $e = (H, \vec{\theta})$ is an example then the count of (F, Ω) w.r.t. e is $c(F, \Omega, e) = |\Omega \cap \mathcal{S}(F, e)|$. Here $\Omega \cap \mathcal{S}(F, e)$ is a multi-set containing all the elements from $\mathcal{S}(F, e)$ (with their multiplicities) which are also contained in Ω . Although this definition of *count* works also for features which do not contain any distinguished variables, it is somewhat cumbersome to always write $c(F, \Omega, e)$ when Ω contains just one empty vector. Therefore, we use a shorthand notation $c(F, e) = c(F, \{()\}, e)$ for features without distinguished variables. Similarly, if $\Omega = \{\vec{\theta}\}$, we use a simplified notation $c(F, \vec{\theta}, e) = c(F, \{\vec{\theta}\}, e)$.

Example 9. Let us have a feature

$$F = \text{size}(X, R_1) \wedge \text{edge}(X, Y) \wedge \text{size}(Y, R_2),$$

and an example $e = (H, \vec{\theta})$

$$\begin{aligned} H &= \{\text{size}(a, r_1), \text{size}(b, r_2), \text{size}(c, r_3), \text{edge}(a, b), \text{edge}(b, c)\} \\ \vec{\theta} &= (1, 0, 3) \end{aligned}$$

The feature F and the example e are the same as in Example 7. Let $\Omega = \{(1, 1), (1, 0)\}$ be a set of vectors. Then the count of (F, Ω) w.r.t. the example e is 1 because $|\Omega \cap \mathcal{S}(F, e)| = |\{(1, 0)\}| = 1$.

When using counting features, we are not interested primarily in *probability of features* as we were in Section 4.3, but rather in the distribution of their counts. In fact, we may be often satisfied with moments of these distributions – usually just with mean and variance.

Example 10. Let us have a distribution on learning examples given by $P(H, \Omega) = \int_{\Omega} f_H(\vec{\theta}|H)P(H) d\vec{\theta}$. Let $P(H)$ be the same as in Example 8. Next, let us suppose that we have a feature $F = \text{num}(R_1)$ and a set $\Omega = [0, 0.1]$ (notice that we use a different set Ω in this example). We compute the mean of the pair (F, Ω) conditioned on the structure of the example and get the following results

$$\begin{aligned} \mathbf{E}(c(F, \Omega, (H, \vec{\theta}))|H = \{\text{num}(r_1)\}) &= 0.1 \\ \mathbf{E}(c(F, \Omega, (H, \vec{\theta}))|H = \{\text{num}(r_1), \text{num}(r_2)\}) &= 0.2. \end{aligned}$$

Here, $\vec{\theta}$ is a random variable conditioned on H .

⁴ This is akin to the generalized covers relation for probabilistic ILP [100].

We can see in the above example that as in the case of Boolean features, the expected value of counts may increase with the number of true groundings of a given feature even though the individual continuous random variables are identically and independently distributed.

Counting features are useful on their own for problems where the property of interest depends on the *total* number of true groundings. Using the illustration with prediction of properties of molecules, counting features are useful as local patterns when the properties that we want to predict are determined by the number of certain substructures. On their own, they are not so useful for problems where the property of interest that we want to predict is given by overall *behaviour* of the numerical parameters. Counting features may be used even for this kind of problems when used in global models, e.g. in Markov logic networks.

4.5 POLYNOMIAL FEATURES

Polynomial features represent a class of features which are able to capture regularities in the overall distributions of continuous parameters of learning examples. A polynomial feature may be any feature with at least one distinguished variable R_i . The adjective ‘polynomial’ emphasizes the fact that the result of evaluating a polynomial feature w.r.t. and example is computed as an (average) value of some multivariate polynomial. Unlike for the previous two types of features (Boolean and counting features) we do not define *probability of a feature* for polynomial features. Instead, we define a *sample distribution of a feature*.

Definition 14 (Sample distribution of a feature). *Let us have a distribution on learning examples with the probability function $P(H, \Omega)$ and let F be a feature with at least one distinguished variable. The sample distribution of the feature F is the distribution of random vectors generated by the following process: 1. Draw a random example $e = (H, \vec{\theta})$ from the distribution with the probability function $P(H, \Omega)$. 2. Compute the sample set $S(F, e)$. 3. Draw a vector v randomly and uniformly from $S(F, e)$.*

Example 11. Let us again consider the distribution on learning examples with the probability function $P(H, \Omega)$ from Example 8. Let us have a feature

$$F = \text{num}(R_1).$$

Then the sample distribution of F w.r.t. this distribution is given by the probability density function

$$f_F(\vec{\theta}) = 1 \text{ for } \vec{\theta} \in [0; 1].$$

Notice that $f_F(\vec{\theta})$ does not depend on the structure of sampled examples in this case unlike the probability of the respective Boolean or counting feature. For instance, if we conditioned the samples on the structure $\{\text{num}(r_1), \text{num}(r_2)\}$ we would get the same result.

Note that the random process which defines sample distributions of features does not take into account varying cardinalities of the sample sets. No matter how large a sample set is, only one sample is taken from it. However, as we will see later, all samples from given sample sets can be used for estimation of statistics of the distributions of features.

Now, we turn our attention to polynomial features. We start by defining *monomial features* and their values which will be used to define polynomial features in turn.

Definition 15 (Monomial feature). *A monomial feature M is a pair $(F, (d_1, \dots, d_k))$ where F is a feature with k distinguished variables and $d_1, \dots, d_k \in \mathbf{N}$. Degree of M is $\text{deg}(M) = \sum_{i=1}^k d_i$. Given a non-empty sample set $S(F, e)$, we define the value of a monomial feature $M = (F, (d_1, \dots, d_k))$ w.r.t. example e as*

$$M(e) = \frac{1}{|S(F, e)|} \sum_{\vec{\theta} \in S(F, e)} \vec{\theta}_1^{d_1} \cdot \vec{\theta}_2^{d_2} \cdot \dots \cdot \vec{\theta}_k^{d_k}$$

where $\vec{\theta}_i$ is the i -th component of vector $\vec{\theta}$.

Sometimes, we will also use a more intuitive notation for monomial and polynomial features motivated by the definition of their *value*:

$$(F, (d_1, \dots, d_k)) \equiv^{\text{def}} (F, R_1^{d_1} \cdot R_2^{d_2} \cdot \dots \cdot R_k^{d_k})$$

Monomial features can be used to define *polynomial features* and their values.

Definition 16 (Polynomial feature). *A polynomial feature is an expression of the form $P = \alpha_1 M_1 + \alpha_2 M_2 + \dots + \alpha_k M_k$ where M_1, \dots, M_k are monomial features and $\alpha_1, \dots, \alpha_k \in \mathbb{R}$ (formally expressed as a pair of two ordered sets - one of monomials and one of the respective coefficients). Value of a polynomial feature $P = \alpha_1 M_1 + \dots + \alpha_k M_k$ w.r.t to an example e is defined as $P(e) = \alpha_1 M_1(e) + \alpha_2 M_2(e) + \dots + \alpha_k M_k(e)$. Degree of a polynomial relational feature P is maximum among the degrees of its monomials.*

One reason why polynomial features are useful is that they allow us to express *aggregate* multivariate polynomial functions of the form

$$f(F, e) = \frac{1}{|\mathcal{S}(F, e)|} \sum_{\vec{\theta} \in \mathcal{S}(F, e)} \sum_{j=1}^l \alpha_j \vec{\theta}_1^{d_{j,1}} \cdot \dots \cdot \vec{\theta}_k^{d_{j,k}}$$

where F is an arbitrary feature. It is, for example, possible to have features for computing average values or covariances for features' sample distributions. The next proposition provides a theoretical basis for computation of moments of distributions and similar statistics.

Proposition 3. *Let us fix a distribution on examples with probability function $P(H, \Omega_H)$. Let F be a feature and f_F be the density of its sample distribution. Let $q(\theta)$ be a multivariate polynomial. Let X_i be vectors sampled independently from the distribution with density f_F and let $Y_i = \frac{1}{i} \sum_{j=1}^i q(X_j)$. If Y_i converges in mean to \hat{Y} for $i \rightarrow \infty$ then*

$$\tilde{Y}_i = \frac{1}{i} \sum_{j=1}^i Q(e_j)$$

converges in probability to \hat{Y} for $i \rightarrow \infty$ where Q is a polynomial relational feature given by the polynomial $q(\theta)$ and the feature F , and e_i are independently sampled examples.

Note that the above proposition holds in spite of the fact that the values $Q(e_i)$ are computed from samples (contained in sample sets $\mathcal{S}(F, e_i)$) which need not be independent. The importance of this proposition is that it enables us to estimate higher-order moments of distributions of relational features.

An alternative way to estimate higher-order moments of features' distributions would be to draw randomly just one vector from each sample set and compute the value of the given polynomial features from the obtained set of vectors. Such an alternative approach would have lower power than the approach based on Proposition 3 because the information contained in the unused samples would be simply thrown away (the power would be the same if the sample sets always contained only multiple copies of the same vector, i.e. if the samples in them were extremely dependent, which may also happen sometimes).

Example 12. We might be interested in marginal distribution of *expression-levels* of all pairs of genes which share a common transcription factor. Thus, we could construct a feature

$$F_1 = e(G_1, R_1) \wedge \text{transcriptionFactorOf}(G_1, G_2) \wedge e(G_2, R_2)$$

and then a set of polynomial features based on F corresponding to all the moments that we are interested in. For instance, if we wanted to compute covariance of pairs of genes where one

gene is a transcription factor of the other gene, we would need the feature F_1 and the following features F_2 and F_3 :

$$\begin{aligned} F_2 &= e(G_1, R_1) \wedge \text{transcriptionFactorOf}(G_1, G_2) \\ F_3 &= \text{transcriptionFactorOf}(G_1, G_2) \wedge e(G_2, R_1) \end{aligned}$$

The polynomial features (actually monomial in this case) needed to compute the covariance would be given as

$$\begin{aligned} P_1 &= (F_1, R_1 \cdot R_2) \\ P_2 &= (F_2, R_1) \\ P_3 &= (F_3, R_1) \end{aligned}$$

Finally, the covariance of the pairs of genes where one gene is a transcription factor of the other gene would be computed from a set of examples as follows:

$$c = \frac{1}{m} \sum_{i=1}^m (P_1(e_i) - P_2(e_i) \cdot P_3(e_i)).$$

Notice that the covariance could not be computed using only one polynomial feature.

A convenient property of polynomial relational features is their decomposability. Here, we describe two variants of decomposability.

Proposition 4. *Every polynomial relational feature of degree d can be expressed using monomial features containing at most d distinguished variables.*

Let us illustrate this decomposability-property on an example.

Example 13. Let us have a feature

$$F = a(X, R_1) \wedge e(X, Y) \wedge a(Y, R_2) \wedge e(Y, Z) \wedge a(Z, R_3)$$

and a polynomial feature

$$P = M_1 + M_2$$

where $M_1 = (F, R_1 \cdot R_2)$ and $M_2 = (F, R_1 \cdot R_3)$. Then we can express P also as $P = M'_1 + M'_2$ where $M'_1 = (F_1, R_1 \cdot R_2)$, $M'_2 = (F_2, R_1 \cdot R_2)$ and

$$F_1 = a(X, R_1) \wedge e(X, Y) \wedge a(Y, R_2) \wedge e(Y, Z) \wedge a(Z, W)$$

$$F_2 = a(X, R_1) \wedge e(X, Y) \wedge a(Y, W) \wedge e(Y, Z) \wedge a(Z, R_2).$$

Note that the new features F_1, F_2 differ from F only in that one of the distinguished variables R_i was replaced by an ordinary variable W (i.e. by a variable which does not extract any numerical values).

The decomposability of polynomial features is practical for feature generation because once we set a maximum degree of the polynomial features that we are interested in, we also know that we can use this limit for the number of distinguished variables in the generated features. Therefore it is also sufficient to construct Gaussian features with at most two distinguished variables because mean, variance and covariance may all be computed by combinations of polynomial features of degree at most two.

There is also a second form of decomposability for polynomial features and that is decomposability of *disconnected* features. We say that a formula F is disconnected if it can be rewritten as $F = (J) \wedge (K)$ such that J and K do not have any variable in common (note that the parentheses ensure that also the logical quantifiers are applied to the formulas J and K separately). We use the term *disconnected* also to describe features which are also formulas.

Proposition 5. *Let $F = (J) \wedge (K)$ be a disconnected feature such that J and K do not share any variable. Let $M = (F, (d_1, \dots, d_k))$ be a monomial. Then it is possible to find monomial features M_J and M_K such that $M(e) = M_J(e) \cdot M_K(e)$ for all examples e .*

As a consequence of Proposition 5, we can focus only on constructing connected monomial features which also means that we have to search a much smaller space of features. A perhaps more important consequence of this proposition is that it allows us to develop an algorithm for evaluating bounded-treewidth polynomial features in time polynomial in the number of their distinguished variables which is described in Chapter 6.

4.6 GENERALIZED MULTIVARIATE-AGGREGATION FEATURES

Polynomial relational features can be generalized straightforwardly. It suffices to replace the multivariate polynomials by other functions. However, then most properties of polynomial relational features no longer hold. For example, the decomposability property for disconnected may no longer be valid. More importantly, the complexity of evaluating generalized multivariate aggregation features may be much higher than the complexity of evaluating polynomial features even when the features have the same structure and differ only by the used aggregation function. In Chapter 6, we show that polynomial features can be evaluated in time polynomial in the combined size of the feature, example and the polynomial when the underlying logical formula is treelike or has bounded treewidth. On the other hand, the problem of evaluating a generalized multivariate feature may be NP-hard even for very simple formulas with bounded treewidth and very simple aggregation functions.

Proposition 6. *The problem of computing*

$$V = \frac{1}{|\mathcal{S}(F, e)|} \sum_{\vec{\theta} \in \mathcal{S}(F, e)} B(\vec{\theta}_1, \vec{\theta}_2, \dots, \vec{\theta}_k)$$

where $B(x_1, x_2, \dots, x_k)$ is a Boolean formula and F is a feature with treewidth 1, is NP-hard.

One of the reasons why we will be able to show in Chapter 6 that polynomial relational features can be evaluated in polynomial time for bounded-treewidth features is that the polynomials in them are represented in the expanded form. If they were allowed to be represented arbitrarily (e.g. $x_1 \cdot x_2 + (x_1 - x_2)^3$) then the problem would be NP-hard too even for bounded-treewidth features.

Proposition 7. *The problem of computing*

$$V = \frac{1}{|\mathcal{S}(F, e)|} \sum_{\vec{\theta} \in \mathcal{S}(F, e)} P(\vec{\theta}_1, \vec{\theta}_2, \dots, \vec{\theta}_k)$$

where $P(x_1, x_2, \dots, x_k)$ is a multivariate polynomial represented as an arbitrary algebraic expression involving only summation, multiplication and integer-exponentiation, and F is a feature with treewidth 1, is NP-hard.

Multivariate aggregation is therefore possible also with different aggregation functions than multivariate polynomials but the computational complexity may be much higher than for the polynomial aggregation functions even if the features are the same and the aggregation functions themselves are efficiently computable (as is the case for Boolean formulas).

4.7 RELATED WORK

As we have already mentioned, what we term *Boolean features* is the most common type of features used in classical inductive logic programming. Counting features are used in relational

learning less frequently on their own but they are used implicitly in approaches like Markov logic [28]. Counting features were also used in the system RELAGGS [58] which is based on univariate relational aggregation. Although we are not aware of any work where polynomial features would be used, they may be also implicitly used in hybrid Markov logic which is a generalization of Markov logic to hybrid domains⁵.

There is, in fact, a tight relationship between hybrid Markov logic [132] and polynomial relational features. Hybrid Markov logic is an extension of Markov logic to hybrid domains, i.e. to domains which contain both discrete relational data and numerical data. A hybrid Markov logic network is a set of pairs (F_i, w_i) where F_i is a first-order formula or a numeric term and $w_i \in \mathbb{R}$. When one fixes a set of constants C then hybrid Markov logic defines a probability distribution over possible worlds as follows:

$$P(X = x) = \frac{1}{Z} \exp \left(\sum_i w_i s_i(x) \right)$$

where $s_i(x)$ is either the number of true groundings of feature F_i w.r.t x if F_i is not a numeric-feature, or the sum of *values* w.r.t. x when F_i is a numeric feature. If one has conditional distributions represented as hybrid Markov logic networks for two classes and the task is to learn a classifier distinguishing examples from the two classes then the resulting decision boundary has equation (where $+$ and $-$ superscripts distinguish the weights and features of the models for the two classes)

$$\sum_i w_i^+ s_i^+(x) - \sum_i w_i^- s_i^-(x) = t$$

which defines a linear hyperplane. This form has already been exploited in the work on max-margin learning of Markov logic networks [51]. If one used polynomial numeric features in the hybrid Markov logic network, the resulting classifier would become very similar to what we obtain with polynomial relational features and a linear classifier, e.g. support vector machine, the main difference being that $s_i(x)$ would be an *average* of values of polynomials applied on the ground instances of feature F_i whereas it is a *sum* in the case of hybrid Markov logic. A convenient property of averages used in polynomial-feature framework is that they are not so sensitive to the size of the examples (possible worlds). The polynomial relational features can be used in conjunction with almost any attribute-value classifier, not just the linear ones, thus possibly offering greater flexibility in discriminative classification.

Polynomial features are also related to aggregation-based approaches to relational learning. The main difference of features used in these systems, e.g. RELAGGS [58], and the polynomial features presented here is that polynomial features allow multivariate aggregation whereas all the existing systems, we are aware of, use only univariate aggregation. A system that could potentially use multivariate aggregation is the system described in [129]. As it uses complex aggregates using Prolog and therefore it should not be hard to define multivariate aggregation in it; however, another question is scalability of such an extension of this system.

4.8 CONCLUSIONS

In this chapter, we described several types of relational features. Our exposition was directed towards hybrid domains. We demonstrated the differences between the several types of features – specifically, what influences their values. We have also briefly touched complexity of evaluating them. In the subsequent chapters, we present algorithms for construction and evaluation of these types of features. In Chapter 5, we describe two fast algorithms for exhaustive construction of a class of Boolean features. In Chapter 6, we describe an algorithm for exhaustive construction of a class of counting and polynomial features. There we also show that polynomial

⁵ In the end, almost any real-valued function can be used in hybrid Markov logic due to the generality with which they were defined.

features with bounded treewidth may be evaluated in polynomial time (recall that in this chapter, we have shown that evaluating generalized multivariate aggregation features with bounded treewidth is NP-hard). Then, in Chapter 7, we present an algorithm for construction of large complex Boolean features. In experiments with real-life bioinformatics problems described in Chapters 9 and 10, we show how counting features can be used for prediction of functions of biological (macro)molecules. In chapters where we do not deal explicitly with problems involving real-valued parameters, we do not represent learning examples as pairs $(H, \vec{\theta})$. We represent them either as Herbrand interpretations or, in some other cases, as clauses. This is either stated explicitly in the respective chapters or follows implicitly from the context.

4.9 PROOFS

This section contains proofs of propositions that do not rely on the rigorous definition of the probabilistic framework. The only proposition which requires treatment in the rigorous framework (Proposition 3 is postponed to Appendix).

Proposition 4. *Every polynomial relational feature of degree d can be expressed using monomial features containing at most d distinguished variables.*

Proof. Clearly, in order to compute a value of any monomial M of order d we need at most d distinguished variables because at most d of them can have non-zero exponent in the monomial. If we replace the distinguished variables with zero d_i in M by ordinary variables (i.e. those that do not extract any numerical values) then the new sample-set which we obtain for an example e and the new feature with fewer distinguished variable will differ from the original sample-set only in that every vector in it will miss the entries associated to the removed distinguished variables. These variables are not used in the computation of values of M anyway. \square

Proposition 5. *Let $F = (J) \wedge (K)$ be a disconnected feature such that J and K do not share any variable. Let $M = (F, (d_1, \dots, d_k))$ be a monomial. Then it is possible to find monomial features M_J and M_K such that $M(e) = M_J(e) \cdot M_K(e)$ for all examples e .*

Proof. It holds $\mathcal{S}(F, e) = \mathcal{S}(J, e) \times \mathcal{S}(K, e)$ where \times denotes Cartesian product. We can therefore construct the monomial features as follows. First, we split the set of distinguished variable of F to two (necessarily disjoint) sets $\mathcal{R}_J, \mathcal{R}_K$ according to the formula in which they appear. We split also the respective exponents to ordered sets (v_1, \dots, v_{k_J}) and (w_1, \dots, w_{k_K}) .

$$M_J = (J, (d_1^J, \dots, d_{k_J}^J))$$

$$M_K = (K, (d_1^K, \dots, d_{k_K}^K)).$$

The product of these monomial features gives rise to the original feature because

$$\begin{aligned} M_J(e) \cdot M_K(e) &= \left(\frac{1}{|\mathcal{S}(J, e)|} \sum_{\vec{\theta} \in \mathcal{S}(J, e)} \vec{\theta}_1^{v_1} \dots \vec{\theta}_{k_J}^{v_{k_J}} \right) \left(\frac{1}{|\mathcal{S}(K, e)|} \sum_{\vec{\theta} \in \mathcal{S}(K, e)} \vec{\theta}_1^{w_1} \dots \vec{\theta}_{k_K}^{w_{k_K}} \right) = \\ &= \frac{1}{|\mathcal{S}(J, e)| \cdot |\mathcal{S}(K, e)|} \cdot \left(\sum_{\vec{\theta} \in \mathcal{S}(J, e)} \vec{\theta}_1^{v_1} \dots \vec{\theta}_{k_J}^{v_{k_J}} \right) \cdot \left(\sum_{\vec{\theta} \in \mathcal{S}(K, e)} \vec{\theta}_1^{w_1} \dots \vec{\theta}_{k_K}^{w_{k_K}} \right) \\ &= \frac{1}{|\mathcal{S}(J, e) \times \mathcal{S}(K, e)|} \sum_{\vec{\theta} \in \mathcal{S}(J, e) \times \mathcal{S}(K, e)} \vec{\theta}_1^{d_1} \dots \vec{\theta}_k^{d_k} = M(e) \end{aligned}$$

\square

Proposition 6. *The problem of computing*

$$V = \frac{1}{|\mathcal{S}(F, e)|} \sum_{\vec{\theta} \in \mathcal{S}(F, e)} B(\vec{\theta}_1, \vec{\theta}_2, \dots, \vec{\theta}_k)$$

where $B(x_1, x_2, \dots, x_k)$ is a Boolean circuit and F is a generalized multivariate aggregation feature with treewidth 1, is NP-hard.

Proof. We show NP-hardness of the problem by reduction from the #SAT problem [3]. Let $B(x_1, x_2, \dots, x_k)$ be a Boolean formula for which we want to compute the number of satisfiable assignments. Let us construct a feature F and an example e :

$$F = a(R_1) \wedge a(R_2) \wedge \dots \wedge a(R_k)$$

$$e = (\{a(r_1), a(r_2)\}, (0, 1)).$$

Now, F has treewidth 1 and the sample set $\mathcal{S}(F, e)$ contains all Boolean vectors of length k . Therefore

$$V = \frac{1}{|\mathcal{S}(F, e)|} \sum_{\vec{\theta} \in \mathcal{S}(F, e)} B(\vec{\theta}_1, \vec{\theta}_2, \dots, \vec{\theta}_k)$$

is equal to the fraction of satisfying assignments of $B(x_1, x_2, \dots, x_k)$ and $2^k \cdot V$ is the total number of satisfying assignments. \square

Proposition 7. *The problem of computing*

$$V = \frac{1}{|\mathcal{S}(F, e)|} \sum_{\vec{\theta} \in \mathcal{S}(F, e)} P(\vec{\theta}_1, \vec{\theta}_2, \dots, \vec{\theta}_k)$$

where $P(x_1, x_2, \dots, x_k)$ is a multivariate polynomial represented as an arbitrary algebraic expression involving only summation, multiplication and integer-exponentiation, and F is a feature with treewidth 1, is NP-hard.

Proof. We show NP-hardness of this problem by reduction from the #3SAT problem [3]. Let $B(x_1, x_2, \dots, x_k)$ be a 3SAT formula for which we want to compute the number of satisfiable assignments. Any 3SAT formula can be rewritten into a multivariate polynomial in the integer domain by application of the following rules:

- $x_1 \wedge x_2 \rightarrow x_1 \cdot x_2$
- $x_1 \vee x_2 \vee x_3 \rightarrow x_1 + x_2 + x_3 - x_1 \cdot x_2 - x_2 \cdot x_3 - x_1 \cdot x_3 + x_1 \cdot x_2 \cdot x_3$
- $\neg x_1 \rightarrow 1 - x_1$

Unless we expand the polynomial, the transformation can be performed in polynomial-time. Let $P(x_1, \dots, x_k)$ be a multivariate polynomial constructed from the formula B using the rules above. Now, let us construct a feature F and an example e (as we did in the proof of Proposition 6)

$$F = a(R_1) \wedge a(R_2) \wedge \dots \wedge a(R_k)$$

$$e = (\{a(r_1), a(r_2)\}, (0, 1)).$$

Now, F has treewidth 1 and the sample set $\mathcal{S}(F, e)$ contains all 0-1 vectors of length k . Therefore

$$V = \frac{1}{|\mathcal{S}(F, e)|} \sum_{\vec{\theta} \in \mathcal{S}(F, e)} P(\vec{\theta}_1, \vec{\theta}_2, \dots, \vec{\theta}_k)$$

is equal to the fraction of satisfying assignments of $B(x_1, x_2, \dots, x_k)$ and $2^k \cdot V$ is the total number of satisfying assignments. \square

Part II

BLOCK-WISE CONSTRUCTION OF RELATIONAL FEATURES: RELF, HIFI AND POLY

Propositionalization aims at converting structured descriptions of examples into attribute-value descriptions which can be processed by most established machine learning algorithms. A major stream of propositionalization approaches [76, 59, 131] proceeds by constructing a set of features (first-order formulas) which follow some prescribed syntactical constraints and play the role of Boolean attributes. Here we assume that examples are represented as first-order Herbrand interpretations and features are conjunctions of first-order function-free atoms. Feature F acquires value *true* for example I (*covers* the example) if $I \models F$, otherwise it has the *false* value. That is we work with *Boolean features* as defined in Chapter 4.

Features may be viewed as patterns taking the form of relational queries. Thus propositionalization is similar to the framework of *frequent query discovery* where one seeks queries that are frequent in that they hold true for at least a specified minimum number of examples. This framework is represented e.g. by system WARMR [25]. Indeed, propositionalization systems such as RSD [131] also adopt a minimum frequency condition that admissible features must comply with. As well known, the minimum frequency condition is very practical in the popular *level-wise* approach to construct conjunctions. In this approach, followed also by the mentioned systems, each conjunction in level n extends by one atom some conjunction in level $n - 1$. Frequency is *monotone* in that if F is not frequent then $F \wedge a$ is not frequent for any atom a . Thus all descendants of an infrequent query may be safely pruned, substantially adding to the efficiency of the level-wise systems.

The problem is that frequency is arguably not the ideal criterion to assess the quality of features in the propositionalization framework when employed for classification learning. Here, rather than frequent features, one naturally prefers features that well discriminate examples in terms of pre-defined classes. For sakes of pruning, one may want to discard *redundant* features, i.e. those for which another feature providing better discrimination exists. Unfortunately, redundancy is not monotone in the level-wise approach and thus cannot be translated here into a pruning mechanism similar to the one based on frequency. Besides redundancy, another important property of features is *reducibility w.r.t. θ -subsumption*. Given the assumed relational representation, two queries may be semantically equivalent despite their difference in syntax. We have good reason—if only for sakes of interpretability—to discard all *reducible* features, i.e. those equivalent to an existing feature that is syntactically simpler. Reducibility is unfortunately neither monotone in the level-wise approach. For example $p(X)$ is irreducible, $p(X) \wedge p(Y)$ is reducible, and $p(X) \wedge p(Y) \wedge q(X, Y)$ is again irreducible.

The purpose of the work presented in this chapter is to remove these deficiencies by elaborating a novel, *block-wise* approach to construct a feature set by identifying building blocks (smaller conjunctions) out of which all features can be composed. The main assumption on which our algorithm is built is that the features it constructs, when viewed graphically, correspond to hypertrees while blocks correspond to their subtrees. Our feature construction strategy is bottom-up so that the initial set of blocks corresponds to all leaves of possible features. Blocks are then progressively combined together with further connecting atoms into larger blocks and eventually into features.

The main advantage of this approach is that it facilitates monotonicity of the two properties of interest in that features containing a redundant (reducible) block are themselves redundant (reducible). Such blocks are thus immediately and safely discarded. As we also show, the assumption of treelike features would not guarantee monotonicity of the two properties in the more traditional level-wise approach. Thanks to the mentioned assumption, we are able to decide whether a block is reducible in time only quadratic in the size of the block despite the NP-completeness of the reducibility problem for unconstrained conjunctions. As a last advan-

tage, the block-wise approach also allows us to efficiently compute extensions of features in a way similar to the well known *query pack* technique [9]. We describe two algorithms, called RELF and HiFi, based on the above outlined ideas. In experiments, we show our approach to result in both very fast feature construction but also in high classification accuracies.

This chapter is organized as follows. In the next section we define the concepts of templates and features. Templates are used to specify the syntactic bias for features. Section 5.2 shows that the basic conditions we have imposed on templates and features imply the hypertree structure of features. In Section 5.3 we introduce blocks and the *graft* operation through which blocks are combined. Section 5.4 shows how to detect reducible blocks and establishes that reducibility is block-wise monotone. An efficient algorithm for checking reducibility is then described in Section 5.5. In Section 5.6 we formalize the semantics of features through the concepts of domain and extension. Using these concepts, Section 5.7 defines redundancy of features and shows how to detect blocks whose inclusion in any feature will make that feature redundant. In Section 5.8 we synthesize the methodological ingredients into a propositionalization algorithm and show that it is complete in that it generates all useful features complying with the given template. Section 11.3 subjects our algorithm to comparative experimental evaluation in 9 relational classification benchmarks. In Section 5.11 we discuss related work and Section 11.4 concludes the main part of this chapter. There are two additional sections. The first one (Section 5.13) contains all proofs of propositions as well as all lemmas. The second one (Section 5.14) presents results on complexity of construction of features when templates without the conditions imposed in Section 5.2. There we show that deciding whether at least one feature complying with the given language specification exists is an NP-complete problem.

5.1 τ -FEATURES

We will be working with the language of first-order predicate logic free of functions up to constants. Learning examples and features will correspond to interpretations and existentially quantified constant-free atom conjunctions, respectively. We will write first-order logic variables in upper-cases and constants in lower-cases. As the order of atoms in conjunctions will not be important, we will use set-notation even for conjunctions. So expressions such as $|C|$, $a \in C$, $C \subseteq C'$ will be interpreted as if C and C' were sets of atoms. A set of conjunctions is said to be *standardized-apart* if no two conjunctions in the set share a variable. By $\text{vars}(C)$ we denote the set of all variables found in conjunction C . Given a set A of atoms, we denote $\text{args}(A) = \{(a, n) \mid a \in A, 1 \leq n \leq \text{arity}(a)\}$, i.e. $\text{args}(A)$ is the set of all argument places in A . For an atom a , $\text{arg}_i(a)$ is its i -th argument. Atoms a_1 and a_2 in conjunction C are *connected in C* if a_1 shares a term with a_2 or with some atom a_3 of C that is connected to a_2 . Conjunction C is said to be *connected* if every two atoms in C are connected in C . Otherwise C is *disconnected*.

A feature construction algorithm should provide the user with suitable means of constraining the feature syntax. In commonplace inductive logic programming systems such as Progol [86] or Aleph, the syntax is specified through *mode declarations* expressed via dedicated meta-predicates. Mode declarations define the predicates which can be used to build a clause, and assign a *type* and *mode* to each argument of these predicates. Here we propose the notion of a feature *template*, which is equally expressive to mode declarations, yet formally simpler. Besides simplicity, another advantage of feature templates is that compliance therewith is verified through the standard subsumption check. A template will be defined as a set of ground atoms of which all arguments fall in exactly one of two categories ('input' and 'output'). Templates will be further subjected to two restrictions needed to guarantee treelikeness of features that are to be derived from templates.

Definition 17 (Template). *A pre-template is a pair $\tau = (\gamma, \mu)$ where γ is a finite set of ground atoms and $\mu \subseteq \text{args}(\gamma)$. Elements of μ are called input arguments and elements of $\text{args}(\gamma) \setminus \mu$ are called output arguments. We say that τ is a template if (i) every atom in γ has at most one input argument*

and (ii) there is a partial irreflexive order \prec on constants in γ such that $c \prec c'$ if and only if there is an atom a with c in its input argument and c' in its output argument.

A template $\tau = (\gamma, \mu)$ is conveniently represented by writing γ with elements of μ ($\text{args}(\gamma) \setminus \mu$) marked with $+$ ($-$) signs, called *input* (*output*) *modes*. We will use the sign \approx to associate τ with this kind of template representation as in the following example.

Example 14. Let $\tau = (\gamma, \mu)$ where $\gamma = \{\text{hasCar}(c), \text{hasLoad}(c, l), \text{box}(l), \text{tri}(l), \text{circ}(l)\}$ and $\mu = \{(\text{hasLoad}(c, l), 1), (\text{box}(l), 1), (\text{tri}(l), 1), (\text{circ}(l), 1)\}$. Here, τ is a template since there is an order $c \prec l$ complying with Definition 17 and no atom in γ has two input arguments. In the simplified notation we show τ as

$$\tau \approx \text{hasCar}(-c), \text{hasLoad}(+c, -l), \text{box}(+l), \text{tri}(+l), \text{circ}(+l)$$

On the other hand, $\tau_2 \approx a(+x, -y), b(+y, -x)$ is not a template because there is no order that would comply with Definition 17.

We will now define the type of connected conjunctions that comply with a given template τ . The compliance will be witnessed by a substitution θ . We shall distinguish certain roles of variables in such conjunctions; these will be determined by τ and θ .

Definition 18 (τ_θ -conjunction). *Given a template $\tau = (\gamma, \mu)$ a τ_θ -conjunction C is a connected conjunction where all terms are variables and $C\theta \subseteq \gamma$. The occurrence of a variable in the i -th argument of atom a in C is an τ_θ -input occurrence in C if the i -th argument of $a\theta$ is in μ and it is an τ_θ -output otherwise. A variable is:*

- τ_θ -positive in C if it has exactly one τ_θ -input occurrence and no τ_θ -output occurrence
- τ_θ -negative in C if it has exactly one τ_θ -output occurrence and no τ_θ -input occurrence
- τ_θ -neutral in C if it has at least one τ_θ -input and exactly one τ_θ -output occurrence

Example 15. Conjunction $\text{hasLoad}(C, L) \wedge \text{box}(L)$ is a τ_θ -conjunction for τ from Example 14 and $\theta = \{C/c, L/l\}$. Variable C is τ_θ -positive and variable L is τ_θ -neutral in the conjunction.

In what follows, we will abbreviate the phrases ‘ τ_θ -input occurrence’ and ‘ τ_θ -output occurrence’ by the respective words ‘input’ and ‘output.’

Through the condition $C\theta \subseteq \gamma$, the definition effectively requires that variables in τ_θ -conjunction C comply with *types* specified by γ , specifically that each variable in C maps to exactly one type. Features, which we define in turn, are τ_θ -conjunctions which additionally comply to a condition pertaining to *modes*.

Definition 19 (τ -Feature). *A τ_θ -conjunction F is a τ_θ -feature if all its variables are τ_θ -neutral in F . We say that F is a τ -feature if there exists a substitution θ such that F is a τ_θ -feature.*

For example, using τ from Example 14, the following conjunction is a τ -feature

$$\text{hasCar}(C) \wedge \text{hasLoad}(C, L1) \wedge \text{hasLoad}(C, L2) \wedge \text{box}(L1) \wedge \text{tri}(L2)$$

as it is a τ_θ -feature for $\theta = \{C/c, L1/l, L2/l\}$. In general, there can be more than one substitution θ such that $F\theta \subseteq \gamma$, and some of them can make some of the variables non-neutral. For example, if

$$\tau = \text{atom}(-a), \text{bond}(+a, -b), \text{bond}(-c, +b), \text{atom}(+c)$$

and

$$F = \text{atom}(A), \text{bond}(A, B), \text{bond}(C, B), \text{atom}(C)$$

then F is a τ_{θ_1} -feature for $\theta_1 = \{A/a, B/b, C/c\}$ but it is not τ_{θ_2} -feature for $\theta_2 = \{A/c, B/b, C/c\}$. In any case, F is a τ -feature because of the existence of θ_1 . It would be easy to avoid the illustrated

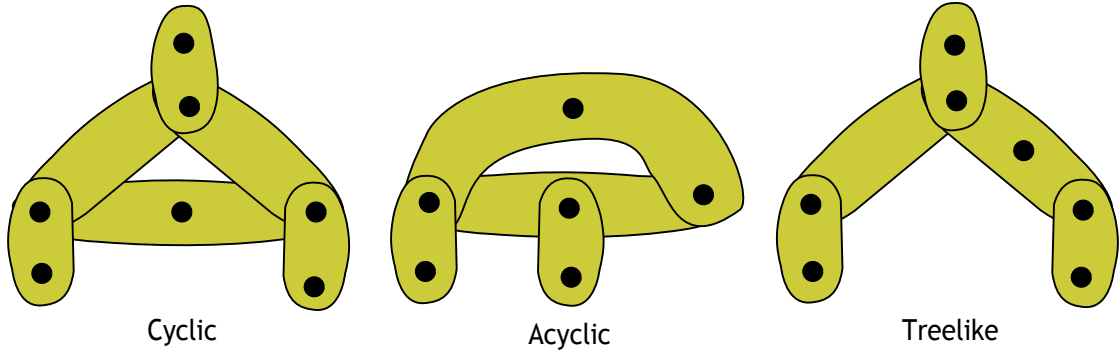


Figure 1: Examples of cyclic, acyclic and treelike hypergraphs that represent first-order conjunctions.

ambiguity of templates, e.g. by requiring that no predicate occurs twice in a template. However, we do not need this restriction.¹

Templates introduced in this section are similar to WARMR’s warmode declarations but not the same. In a template, every atom can have at most one input argument, unlike warmodes which do not restrict the number of input or output arguments. We do need the restriction to guarantee treelike structure of features. Furthermore, variables in our features must have both an input and an output. This requirement increases the expressiveness of our framework (the case without the requirement can clearly be emulated by adding dummy atoms). Further justification for the requirement is given in [76].

5.2 TREELIKE STRUCTURE OF τ -FEATURES

The two conditions we imposed on templates in Definition 17 and the neutrality condition we stipulated on variables in features in Definition 19 guarantee a constrained structure of features when viewed graphically. For precise characterization, we shall define treelikeness of conjunctions which is a special case of acyclicity. We follow the established definition from [29] that relates to hypergraphs. While we translate the notions into the context of conjunctions, the definition also shows the original hypergraph concepts in parentheses since hypergraphs offer easier intuitive understanding.

Definition 20 (Treelike conjunction). *A conjunction (hypergraph) C is treelike if the iteration of the following rules on C produces the empty conjunction (hypergraph):*

1. Remove an atom (hyperedge) which contains fewer than 2 terms (vertices).
2. Remove a term (vertex) which is contained in at most one atom (hyperedge).

The right-most panel in Figure 1 illustrates the concept of treelikeness. A few remarks are needed to fully clarify the correspondence between conjunctions and hypergraphs. Vertices correspond to terms and hyperedges correspond to atoms that contain the terms as arguments. In general, multiple atoms in a conjunction may contain the same set of terms. Therefore conjunctions correspond in fact to *multi-hypergraphs*. This fact is not important when applying the iterative reduction above and thus for brevity we maintain the shorter term *hypergraph*. In Definition 38, removing a term from C means removing all its occurrences from C . That is, the arity of each atom in C will be decreased by the number of occurrences of the term in the atom.

Obviously, general τ_θ -conjunctions may be non-treelike. For example,

$$C = a(A, B) \wedge a(C, B) \wedge a(C, D) \wedge a(A, D)$$

¹ In fact, with certain abstraction, this ambiguity is analogical to that present in inductive logic programming systems where different sequences of refinements may result in the same first-order expression.

is a τ_θ -conjunction for $\tau = (\gamma, \mu)$, $\gamma = \{a(a, b)\}$, $\theta = \{A/a, B/b, C/a, D/b\}$ and any $\mu \subseteq \text{args}(\gamma)$. C is not treelike as may be verified by applying Definition 38 (clearly, none of the two reduction rules can be applied on C) or by projecting C onto the left-most example in Figure 1. However, as the following proposition asserts, τ -features are indeed treelike.

Proposition 8. *Every τ -feature F is treelike.*

A convenient consequence is that the decision problem $I \models F$ can be computed in time polynomial in $|F|$. In contrast, for a general conjunction C , $I \models C$ corresponds to the NP-complete θ -subsumption test. Polynomial-time solubility of treelike queries is well known in database literature [137] and in the field of constraint satisfaction where this problem is efficiently tackled by the directional-arc-consistency algorithm [24].

5.3 BLOCKS

Here we first define *blocks* used to build a feature through the *graft* operation². Before we do so formally, we motivate the concepts through an example. Given a template

$$\tau \approx \text{hasCar}(-c), \text{hasLoad}(+c, -l), \text{has2Loads}(+c, -l, -l), \text{box}(+l), \text{tri}(+l)$$

an exemplary τ_θ -feature for $\theta = \{C/c, L1/l, L2/l, L3/l\}$

$$\text{hasCar}(C) \wedge \tag{2}$$

$$\text{hasLoad}(C, L1) \wedge \text{box}(L1) \wedge \tag{3}$$

$$\text{has2Loads}(C, L2, L3) \wedge \text{box}(L2) \wedge \text{box}(L3) \tag{4}$$

is graphically shown left-most in Fig. 2 as a hypertree containing two subtrees corresponding to lines 2 and 3 above. The latter subtree is also shown separately in the middle of Fig. 2. Conjunctions corresponding to such subtrees will be called *pos τ_θ -blocks* to abbreviate the fact that they contain exactly one τ_θ -positive variable (here C). This is the only variable that the block shares with the rest of the feature. The share of only one variable has a convenient consequence towards monotonicity, as we will also formally show: if the block is redundant (reducible), the containing feature is also redundant (reducible).

The second kind of block we shall define corresponds to what is left of a τ_θ -feature when a *pos τ_θ -block* is removed. Such a block, called a *neg τ_θ -block* contains exactly one τ_θ -negative variable. An example of a *neg τ_θ -block* is shown right-most in Fig. 2.

Definition 21 (Blocks). *Let B be a τ_θ -conjunction. If there is exactly one τ_θ -positive (τ_θ -negative) variable in B , it will be denoted $p(B)$ ($n(B)$). B is a *pos τ_θ -block* if it has $p(B)$ ($n(B)$) and all other variables in B are τ_θ -neutral. We say that B is a *pos (neg) τ -block* if there exists a substitution θ such that B is a *pos (neg) τ_θ -block*.*

Of course, a *pos τ_θ -block* can itself in general contain smaller *pos τ_θ -blocks*. To exploit the above commented monotonicity observed on blocks, our strategy to assemble features will be *bottom-up*, starting with the smallest *pos τ_θ -blocks* (e.g. $\text{tri}(L3)$ in the current example) and safely discarding those deemed redundant or reducible. Larger *pos τ_θ -blocks* result from conjoining a number of smaller τ_θ -blocks (e.g. $\text{box}(L2)$ and $\text{tri}(L3)$) with further atoms needed to make the conjunction a *pos τ_θ -block* (in this case $\text{has2Loads}(C, L2, L3)$) or a τ_θ -feature. This operation, which we call a *graft*, is formalized below.

Definition 22 (Graft). *Let C be a τ_θ -conjunction, v a variable in C , and $\phi^+ = \{B_i^+\}$ a standardized-apart set of *pos τ -blocks*. We define $C \oplus_v \phi^+ = C \wedge_i B_i \theta_i$ where $\theta_i = \{p(B_i^+)/v\}$. In the special case when C is a *neg τ -block*, we denote $C \oplus \phi^+ = C \oplus_{n(C)} \phi^+$.*

² not to be confused with grafting as introduced in [95]

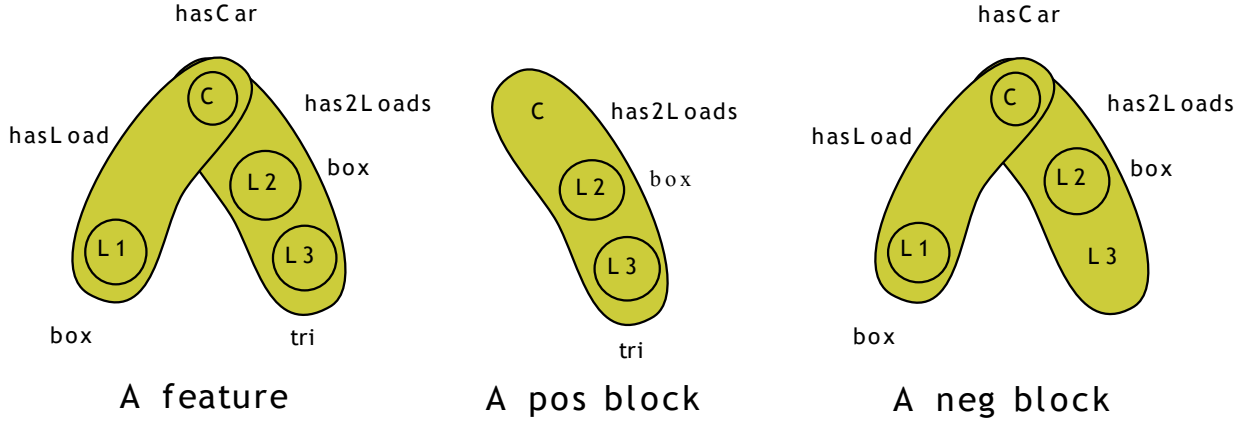


Figure 2: A feature, a pos block and a neg block displayed as hypergraphs

In the mentioned special case when C is a neg τ -block, $C \oplus \phi^+$ is a τ_θ -feature. Otherwise, for generality, the definition does not require that the graft results in a pos τ -block. Speaking in terms conventional for inductive logic programming, graft is an operator yielding a common specialization of its operands. However, in comparison with well known operators such as Plotkin’s least general generalization [92], graft is defined in a highly constrained context of treelike function-free conjunctions rather than general first-order clauses.

5.4 REDUCIBILITY

In this section, we elaborate properties of features and pos blocks regarding their reductions. Let C and D be conjunctions of atoms and let there be a substitution θ such that $C\theta \subseteq D$. Then we say that C *subsumes* D (written $C \preceq D$). If $C \preceq D$ and $D \preceq C$ then C and D are said to be *equivalent*, written $C \approx_\theta D$. We say that C is *reducible* if there exists a conjunction C' such that $C \approx_\theta C'$ and $|C| > |C'|$; C' is called a *reduction* of C . An example of a reducible conjunction of atoms is $\text{hasCar}(C) \wedge \text{hasLoad}(C, L1) \wedge \text{hasLoad}(C, L2) \wedge \text{box}(L1)$ equivalent to the shorter conjunction $\text{hasCar}(C) \wedge \text{hasLoad}(C, L1) \wedge \text{box}(L1)$.

In this work we cannot rely directly on the established notion of reducibility as defined above. This is because a reduction of a τ -feature may not be a τ -feature itself. For example, for $\tau \approx \{\text{car}(-c_1), \text{hasLoad}(+c_1, -l), \text{hasLoad}(-c_2, +l), \text{hasCar}(+c_2)\}$, the conjunction $\text{car}(C_1) \wedge \text{hasLoad}(C_1, L_1) \wedge \text{hasLoad}(C_2, L_1) \wedge \text{car}(C_2)$ is a correct τ -feature but its reduction $\text{hasCar}(C_1) \wedge \text{hasLoad}(C_1, L_1)$ is not. The fact that reduction does not preserve correctness of feature syntax may represent a problem because we would like to work only with reduced features, if only for sakes of improved interpretability. Relying on the treelikeness of features and the implied treelikeness of pos blocks, we are going to introduce H-subsumption and H-reduction such that H-reduction of a τ_θ -feature is always a τ_θ -feature. The two concepts differ from their classical counterparts by requiring that substitutions involved in the subsumption relations preserve variable *depths*. We define the notion of depth in turn.

Definition 23 (Depth). *Let C be a τ_θ -feature or a pos τ -block and a its atom. If a contains no inputs or if it contains the variable $p(C)$ then its depth in C , denoted $d_C(a)$ is 1. Otherwise $d_C(a) = n$ such that a_1, a_2, \dots, a_n is the shortest sequence of atoms such that $d_C(a_1) = 1$, $a_n = a$ and for each $1 \leq i < n$, there is a variable having an output in a_i and an input in a_{i+1} . The depth of variable v in C is $d_C(v) = \min\{d_C(a) \mid a \text{ contains } v\}$.*

Definition 24 (H-subsumption). *We say that τ_{θ_1} -feature (pos τ -block, respectively) F H-subsumes τ_{θ_2} -feature (pos τ -block, respectively) G (written $F \preceq_H G$) if and only if there is a substitution ϑ such that $F\vartheta \subseteq G$ and for every atom $a \in F$ it holds $d_F(a) = d_G(a\vartheta)$; such a ϑ is called a H-substitution.*

Definition 25 (H-reduction). If $F \preceq_H G$ and $G \preceq_H F$, we call F and G H-equivalent (written $F \approx_H G$). We say that τ_{θ_1} -feature (pos τ -block, respectively) F is H-reducible if there is a τ_{θ_2} -feature (pos τ -block, respectively) F' such that $F \approx_H F'$ and $|F| > |F'|$. A τ_{θ} -feature (pos τ -block) F' is said to be an H-reduction of F if $F \approx_H F'$ and F' is not H-reducible. A τ -feature F is said to be H-reducible if there is a substitution θ such that F is an H-reducible τ_{θ} -feature.

Example 16. Continuing with τ from Example 14, the τ -feature

$$F_1 = \text{hasCar}(C) \wedge \text{hasLoad}(C, L1) \wedge \text{box}(L1) \wedge \text{hasLoad}(C, L2) \wedge \text{box}(L2) \wedge \text{tri}(L2)$$

is H-reducible because there is feature $F_2 = \text{hasCar}(C) \wedge \text{hasLoad}(C, L) \wedge \text{box}(L) \wedge \text{tri}(L)$ for which it holds $F_1 \approx_H F_2$, as $F_1\theta_1 \subseteq F_2$ and $F_2\theta_2 \subseteq F_1$ for substitutions $\theta_1 = \{C/C, L1/L, L2/L\}$, $\theta_2 = \{C/C, L/L2\}$, which map variables in depth d in one feature to variables in the same depth in the other feature, and it holds $|F_2| < |F_1|$.

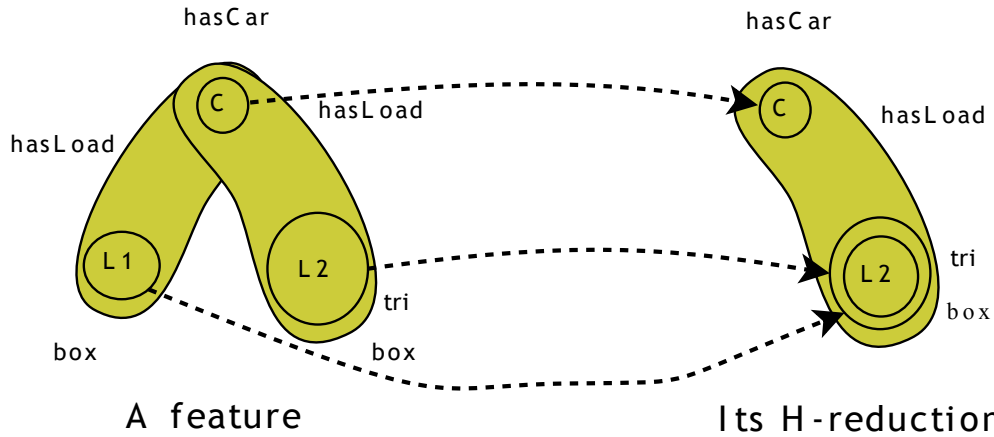


Figure 3: The feature F from Example 16 (left) and its H-reduction (right); arrows indicate the substitution θ_1 .

Since H-reducibility is the only concept of reducibility we will work with, we will occasionally afford to omit the H- prefix. The next proposition provides a way to detect reducibility of pos blocks and also asserts that reducibility is monotone.

Proposition 9. Let B be a pos τ -block and let B^- be a neg τ -block. Then the following holds: (i) B is H-reducible if and only if B contains pos τ -blocks B_1, B_2 such that $B_1 \neq B_2$, $p(B_1) = p(B_2)$ and $B_1 \preceq_H B_2$. (ii) If B is H-reducible, then $B^- \oplus (\{B\} \cup S)$ is also H-reducible for any set of pos τ -blocks S .

Monotonicity of reducibility is an important property enabling pruning during feature construction. It is essential to realize that this property does not simply follow from the treelike structure of features. Indeed, if we constructed treelike features in the usual level-wise manner as e.g. the systems WARMR or RSD would proceed, it would not be possible to prune on reducibility. For example, extending $\text{hasCar}(C) \wedge \text{hasLoad}(C, L) \wedge \text{box}(L)$ by atom $\text{hasLoad}(C, L2)$ yields a reducible conjunction. If we however pruned it along with all of its descendants, we would also prune the irreducible feature $\text{hasCar}(C) \wedge \text{hasLoad}(C, L) \wedge \text{box}(L) \wedge \text{hasLoad}(C, L2) \wedge \text{tri}(L2)$.

Lastly, thanks to the reducibility concept, we can state the following proposition which is important since we aim at constructing *complete* feature sets.

Proposition 10. Let τ be a template. There is only a finite number of τ -features, which are not H-reducible.

This holds despite that there is an infinite number of τ -features for a sufficiently rich template τ .

Example 17. Let us work again with τ from Example 14. There is an infinite number of τ -features but there are only 19 irreducible τ -features. An example of a reducible τ -feature is

$$F_\tau = \text{hasCar}(C) \wedge \text{hasLoad}(C, L) \wedge \text{box}(L) \wedge \text{hasLoad}(C, L2) \wedge \text{box}(L2) \wedge \text{circ}(L2)$$

This τ -feature is indeed reducible, which according to Proposition 9 follows from the presence of the two pos blocks below in it.

$$\text{hasLoad}(C, L) \wedge \text{box}(L) \preceq_H \text{hasLoad}(C, L2) \wedge \text{box}(L2) \wedge \text{circ}(L2)$$

As may be routinely checked, there are exactly eighteen further irreducible τ -features.

5.5 A FAST ALGORITHM FOR H-REDUCTION

We now describe an algorithm for checking H-reducibility of features (Algorithm 1) that runs in time quadratic in the number of atoms in the tested feature. We need to introduce some additional notation which will be constrained to this section. Let F be a τ_θ -feature or a pos τ_θ -block and P_F, I_F, J_F , functions defined as follows. If two atoms $a, b \in F$ share a variable, which has an output in the i -th argument of a and an input in the j -th argument of b we define $P_F(b) = a, I_F(b) = i, J_F(b) = j$. Lastly, for an atom b in F we define $C_F(b) = \{a \in F \mid P_F(a) = b\}$. That is, $P_F(a)$ denotes the ‘parent’ atom of a , which, due to the treelike structure of features and pos blocks, is unique if it exists. Inversely, $C_F(b)$ is the set of all ‘children’ atoms of b . Functions I_F and J_F index the arguments in the respective literals where the ‘connecting variable’ occurs.

The algorithm is based on Proposition 9, namely on the fact that a τ_θ -feature F is H-reducible if and only if it contains two pos τ -blocks B_1 and B_2 such that $p(B_1) = p(B_2)$ and $B_1 \preceq_H B_2$. The main idea exploited by the algorithm is to maintain an array called *Subsumed* indexed by pairs (α_i, α_j) in which we store the number of child-blocks (i.e. pos τ -blocks) of the atom α_i , which H-subsume some child-blocks of the atom α_j such that $I_F(\alpha_i) = I_F(\alpha_j)$. If this number equals the number of children of α_i and if α_i and α_j share a common parent, then the tested τ -feature F is H-reducible. It is easiest to see how this algorithm works through an example.

Example 18. Consider template $\tau \approx a(-x), b(+x, -y), c(+y), d(+y)$ and the following feature

$$F = a(A) \wedge b(A, B) \wedge c(B) \wedge d(B) \wedge b(A, C), c(C)$$

Let us go through the steps Algorithm 1 performs to detect that F is H-reducible. It starts by filling the *Open* list: $\text{Open} \leftarrow \{(c(B), c(C)), (c(C), c(B))\}$. In the next step, the first pair $(c(B), c(C))$ is extracted from the list and we check whether $P_F(c(B)) = P_F(c(C))$. As the answer is negative, the algorithm continues by incrementing $\text{Subsumed}(b(A, B), b(A, C))$, i.e. we have $\text{Subsumed}(b(A, B), b(A, C)) = 1$. However, $\text{Subsumed}(b(A, B), b(A, C)) \neq |C_F(b(A, B))| = 2$, so we continue with the next iteration. We extract $(c(C), c(B))$ from the list *Open*. Again $P_F(c(B)) \neq P_F(c(C))$, therefore we increment $\text{Subsumed}(b(A, C), b(A, B))$. We have $\text{Subsumed}(b(A, C), b(A, B)) = |C_F(b(A, C))| = 1$, therefore we add $(b(A, C), b(A, B))$ to the *Open* list. In the next iteration, we extract $(b(A, C), b(A, B))$ from the list *Open* and check that $P_F(b(A, C)) = P_F(b(A, B))$. From this, we may conclude that F is H-reducible.

Proposition 11. *Algorithm 1 correctly decides whether a τ_θ -feature (pos τ -block) F is H-reducible in time $O(|F|^2)$.*

Algorithm 1, which only decides whether a τ -feature F is H-reducible, can be easily converted to an algorithm that also computes the H-reduction³ of F . It suffices to replace line 10 in Algorithm 1 by ‘Remove pos τ -block with root α_i ’.

³ The basic principles of the algorithm can be also used to easily obtain an $O(|F| \cdot |G|)$ algorithm for deciding $F \preceq_H G$.

Algorithm 1 H-Reducibility(F)

```

1: Input:  $\tau_\theta$ -feature (pos  $\tau$ -block) F;
2: Subsumed  $\leftarrow$  a two-dimensional array indexed by atoms. All elements are initialized to 0. /*
   Subsumed(A, B) is the number of child-blocks of A, which have been found to H-subsume
   some child-blocks of B */
3: Open  $\leftarrow$  {} /* A stack data-structure */
4: for  $d = 1 \dots d_F$  do
5:   Open  $\leftarrow$  Open  $\cup$   $\{(a_i, a_j)\}$  where  $a_i, a_j$  ( $a_i \neq a_j$ ) are literals in depth  $d$  with no outputs
   and such that  $\text{predicate}(a_i) = \text{predicate}(a_j)$  and  $I_F(a_i) = I_F(a_j)$ 
6: end for
7: while Open  $\neq \emptyset$  do
8:    $(a_i, a_j) \leftarrow$  Pop(Open)
9:   if  $P_F(a_i) = P_F(a_j)$  then
10:    return H-Reducible
11:   else
12:     Subsumed( $P_F(a_i), P_F(a_j)$ ) = Subsumed( $P_F(a_i), P_F(a_j)$ ) + 1
13:     if Subsumed( $P_F(a_i), P_F(a_j)$ ) =  $|C_F(P_F(a_i))|$  and  $\text{predicate}(P_F(a_i)) =$ 
        $\text{predicate}(P_F(a_j))$  and  $I_F(P_F(a_i)) = I_F(P_F(a_j))$  and  $J_F(P_F(a_i)) = J_F(P_F(a_j))$  then
14:       Open  $\leftarrow$  Open  $\cup$   $\{(P_F(a_i), P_F(a_j))\}$ 
15:     end if
16:   end if
17: end while
18: return Not H-Reducible

```

5.6 EXTENSIONS AND DOMAINS

So far we have been only concerned with the syntax of τ -features. Now we will also establish their semantics. We do this through the concepts of *domain* and *extension*. Extension of a τ -feature is simply the set of examples covered by the τ -feature. Domain is defined in a finer manner for both τ -features and pos τ -blocks, with respect to a single example. The method for computing domains and extensions will be based on a combination of a well-known algorithm for conjunctive acyclic query answering [137] and a variation of the query-pack method [9], integrated into our block-wise feature construction procedure.

Definition 26 (Domain, Extension). *Let I be an interpretation, i.e. a set of ground atoms, and τ a template. For a pos τ -block B , domain $\text{dom}_I(B)$ contains all terms t such that $I \models B\theta$, where $\theta = \{p(B)/t\}$. For a τ -feature F , $\text{dom}_I(F) = \{\text{yes}\}$ if $I \models F$ and $\text{dom}_I(F) = \emptyset$ otherwise. For a set E of interpretations, the extension of a τ -feature F on E is defined as $\text{ext}_E(F) = \{I \in E \mid I \models F\}$.*

Domain, as defined in Definition 26, assigns to each pos τ -block B a set of terms $\{t_i\}$, for which there is a substitution θ of B such that $p(B\theta) = t_i$ and $B\theta$ is true in I . Note that template τ is not indicated in the subscript of either dom or ext as both of them are independent of τ . This is instantly clear for features. For a pos τ -block B , we must recall from Section 5.1 that $p(B)$ is identified uniquely as it is the only variable with a single occurrence in B . The domain of B may be computed by Algorithm 2 which runs in time polynomial in the size of B . This algorithm is not novel, it corresponds to the algorithm known as conjunctive acyclic query answering algorithm in database theory [137] and also to directed-arc-consistency algorithm in the field of CSP [24].

Example 19. Consider feature F and interpretation I

$$F = \text{hasCar}(C) \wedge \text{hasLoad}(C, L) \wedge \text{tri}(L) \wedge \text{box}(L),$$

$$I = \{\text{hasCar}(c), \text{hasLoad}(c, l1), \text{hasLoad}(c, l2), \text{tri}(l1)\},$$

Algorithm 2 $\text{dom}_I(B)$

```

1: Input: Pos  $\tau$ -block  $B$ , Interpretation  $I$ ;
2:  $\text{atomsDom} \leftarrow \{a \in I \mid \text{pred}(a) = \text{pred}(\text{root}(B)) \wedge (I \models a)\}$ 
3: for  $\forall$  output variables  $\text{out}_i$  of  $B$ 's root do
4:    $\text{Children} \leftarrow$  all pos blocks  $B$  such that  $\text{out}_i = p(B)$  (i.e. children of  $B$ 's root hanging on
      its  $i$ -th output)
5:    $\text{argDom}_i \leftarrow \bigcap_{c \in \text{Children}} \text{dom}_I(c)$ 
6:    $\text{atomsDom} \leftarrow \text{atomsDom} \cap \{a \in I \mid \text{arg}_i(a) \in \text{argDom}_i\}$ 
7: end for
8: return  $\{t \mid t \text{ is term at input of an atom } a \text{ and } a \in \text{atomsDom}\}$ 

```

$\text{circ}(l1), \text{tri}(l2), \text{box}(l2), \text{hasLoad}(c, l3), \text{box}(l4)\}$.

We may proceed as follows: (i) We compute domains of pos τ -block $\text{box}(L)$ and $\text{tri}(L)$, i.e. $\text{dom}_I(\text{box}(L)) = \{l2, l4\}$, $\text{dom}_I(\text{tri}(L)) = \{l1, l2\}$. (ii) Then we compute domain of pos τ -block with root $\text{hasLoad}(C, L)$, i.e. $\text{dom}_I(\text{hasLoad}(C, L) \wedge \text{tri}(L) \wedge \text{box}(L)) = \{l1, l2, l3\} \cap \{l2, l4\} \cap \{l1, l2\} = \{l2\}$. The first set in the intersection comes from the fact that the predicate symbol is $\text{hasLoad}/2$. The second set is due to pos τ -block $\text{box}(L)$ and the third set is due to pos τ -block $\text{tri}(L)$. (iii) Since no domain is empty so far, we proceed further and compute domain of the feature with root $\text{hasCar}(C)$, which becomes $\{\text{yes}\}$. So, we see that $I \models F$.

The example makes it clear that computation of domains can be combined with the earlier described block grafting in a way resembling query packs [9]. When a pos block is constructed grafting an already constructed block c , the domain previously computed for c is reused as $\text{dom}_I(c)$ in line 5 of Algorithm 2. This principle is further illustrated in Fig. 4. To prevent excessive memory demands possibly resulting from the storing of domains, our implementation uses a Java mechanism enabling it to automatically discard computed domains when a memory limit is reached.

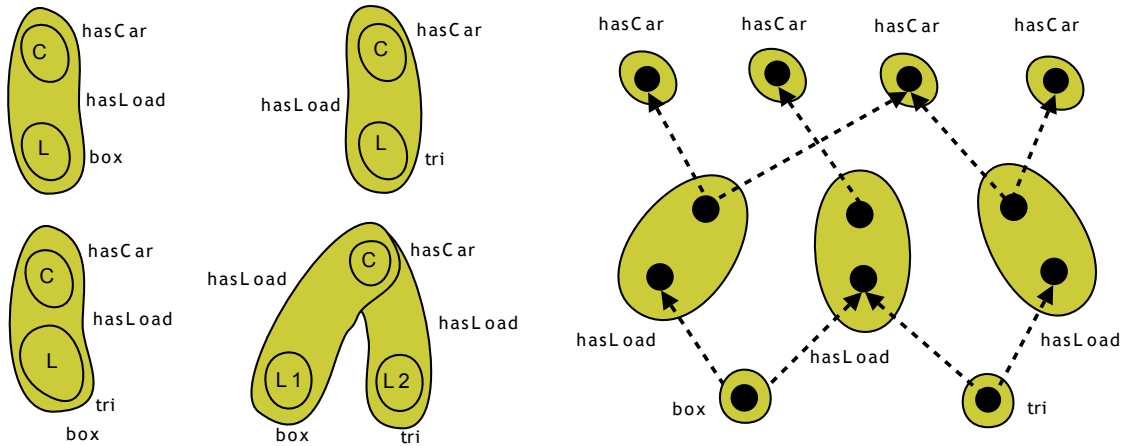


Figure 4: **Left:** The set of all four non-H-reducible τ -features for $\tau \approx \text{hasCar}(-c), \text{hasLoad}(+c, -l), \text{box}(+l), \text{tri}(+l)$. **Right:** the structure of domain reusing when computing domains of these features. The arrows indicate the flow of domain information. If domains are reused, only 9 domains need to be computed, whereas, if features were treated in isolation from each other, 15 domains would need to be computed.

5.7 REDUNDANCY

We shall now define when a feature is redundant. We do so in the spirit of [77] where redundancy was termed *irrelevancy*. Redundancy of a feature will be defined with respect to a feature

set \mathcal{F} and two sets of examples (interpretations) E^+ and E^- although, for brevity, we will not explicitly say ‘with respect to \mathcal{F} , E^+ and E^- ’ when speaking of redundancy concepts. Adopting these concepts, our approach becomes limited to binary classification problems, including of course the case where a single target class is to be discriminated from a number of other classes.

Definition 27 (Redundancy). *Let \mathcal{F} be a set of τ -features for some template τ . Let E^+ and E^- be two sets of interpretations (the positive and negative examples, respectively). F is E^+ -redundant (E^- -redundant) if there is $F' \in \mathcal{F}$, $F \neq F'$ such that $\text{ext}_{E^+}(F) \subseteq \text{ext}_{E^+}(F')$ and $\text{ext}_{E^-}(F') \subseteq \text{ext}_{E^-}(F)$ ($\text{ext}_{E^-}(F) \subseteq \text{ext}_{E^-}(F')$ and $\text{ext}_{E^+}(F') \subseteq \text{ext}_{E^+}(F)$). If one of the inclusions, for at least one example, is strict, F is said to be strictly E^+ -redundant (E^- -redundant). F is called redundant if it is both E^+ -redundant and E^- -redundant. It is called strictly redundant if it is (i) redundant, and (ii) strictly E^+ -redundant or strictly E^- -redundant.*

	F_1	F_2	F_3	F_4	F_5	Class
Example 1	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus
Example 2	\oplus	\oplus	\ominus	\ominus	\oplus	\oplus
Example 3	\ominus	\oplus	\oplus	\ominus	\ominus	\ominus
Example 4	\oplus	\ominus	\oplus	\oplus	\oplus	\ominus

Table 2: An example set with five features used to illustrate the concept of redundancy in Example 20.

Example 20. Consider the set of examples and features as displayed in Table 2. F_4 is strictly E^+ -redundant because it covers the same negative examples as F_1 but the set of positive examples covered by it is a strict subset of the set of positive examples covered by F_1 . Feature F_4 is also E^- -redundant because it covers the same set of positive examples as F_3 but the set of negative examples covered by F_4 is a strict subset of those covered by F_3 . Therefore F_4 is also strictly redundant. F_1 and F_5 are non-strictly redundant because they cover the same set of examples.

In [77], redundancy was introduced for attribute-value learning and for learning of constrained Horn clauses, which are equivalent in their expressive power to attribute-value representations. Applied to our setting, it is obvious that when features are constructed, they can be filtered in a post-processing step, i.e. strictly redundant features can be discarded and from each non-strictly redundant set of features with equal extensions, one feature can be preserved discarding the rest. What is however less obvious is that a form of redundancy can be detected already for pos blocks and that it implies redundancy of features that contain them. This is formalized in turn.

Proposition 12. *Let E^+ (E^-) be a set of positive (negative) examples. Let \mathcal{F} be a set of pos τ -blocks with equal types of input arguments and let $\mathcal{B} = \{B_i\}_{i=1}^n \subseteq \mathcal{F}$. Let $\text{dom}_I(B_1) \subseteq \bigcap_{i=2}^n \text{dom}_I(B_i)$ for all $I \in E^+$ ($I \in E^-$) and let $\bigcap_{i=2}^n \text{dom}_I(B_i) \subseteq \text{dom}_I(B_1)$ be true for all $I \in E^-$ ($I \in E^+$). Then for any neg τ -block $B^-: B^- \oplus (\{B_1\} \cup \mathcal{S})$ is E^+ -redundant (E^- -redundant), where $\mathcal{S} \subseteq \mathcal{F}$.*

During the generation of features, we will discard pos τ -blocks such as B_1 characterized by the above proposition, i.e. those pos τ -blocks giving rise to redundant features. Let us use the adjective redundant also for such pos τ -blocks. We use the term *monotonicity of redundancy* to denote that once a pos block B is redundant, any block or feature constructed using B is redundant as well. Again, this kind of pruning is enabled by our block-wise strategy and there is no direct analogue to it in the traditional level-wise approach, as was illustrated in Section 5.4 in the context of reducibility.

Example 21. Let us continue with Example 16, considering a set of two positive examples E^+ and a set of two negative examples E^- .

$$E^+ = \{\{\text{hasCar}(c1), \text{hasLoad}(c1, l1), \text{circ}(l1), \text{box}(l1), \text{hasLoad}(c1, l2), \text{tri}(l2)\}\},$$

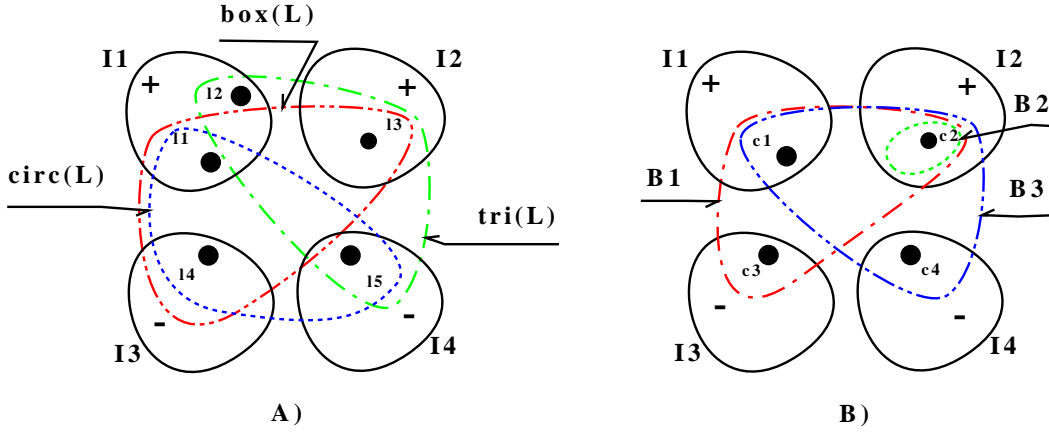


Figure 5: Domains of pos blocks from Example 21. Ovals marked by I_1, I_2 correspond to the positive examples (but not to their hypergraph representation) and ovals marked by I_3, I_4 correspond to the negative examples. The dashed/dotted/dash-dotted ovals correspond to domains of the blocks. The left panel displays domains of blocks $\text{box}(L)$, $\text{tri}(L)$ and $\text{circ}(L)$ and the right panel displays domains of blocks $B_1 = \text{hasLoad}(C, L), \text{box}(L)$, $B_2 = \text{hasLoad}(C, L), \text{box}(L), \text{tri}(L)$ and $B_3 = \text{hasLoad}(C, L), \text{tri}(L)$.

$$\begin{aligned} & \{\text{hasCar}(c2), \text{hasLoad}(c2, l3), \text{box}(l3), \text{tri}(l3)\} \\ E^- = & \{\{\text{hasCar}(c3), \text{hasLoad}(c3, l4), \text{box}(l4), \text{circ}(l4)\}, \\ & \{\text{hasCar}(c4), \text{hasLoad}(c4, l5), \text{tri}(l5), \text{circ}(l5)\}\} \end{aligned}$$

We will construct the E^+ -non-redundant τ -features by block grafting. First, we generate and filter the following three pos τ -blocks: $\text{box}(L)$, $\text{tri}(L)$, $\text{circ}(L)$. We may check that $\text{circ}(L)$ is E^+ -redundant (due to $\text{box}(L)$), so we may safely discard it. The next pos τ -blocks created by composing $\text{box}(L)$ and $\text{tri}(L)$ with $\text{hasLoad}(C', L')$ are pos τ -blocks $B_1 = \text{hasLoad}(C, L) \wedge \text{box}(L)$, $B_2 = \text{hasLoad}(C, L) \wedge \text{box}(L) \wedge \text{tri}(L)$ and $B_3 = \text{hasLoad}(C, L) \wedge \text{tri}(L)$. Notice that if we had not removed $\text{circ}(L)$, there would have been seven such pos τ -blocks. We can now filter also B_1, B_2, B_3 in exactly the same manner as we filtered $\text{box}(L), \text{tri}(L), \text{circ}(L)$. In this case, B_2 is E^+ -redundant because

$$\begin{aligned} \text{dom}_{I_1}(B_2) &= \emptyset \subseteq \text{dom}_{I_1}(B_1) \cap \text{dom}_{I_1}(B_3) = \{c1\}, \\ \text{dom}_{I_2}(B_2) &= \{c2\} \subseteq \text{dom}_{I_2}(B_1) \cap \text{dom}_{I_2}(B_3) = \{c2\}, \\ \text{dom}_{I_3}(B_1) \cap \text{dom}_{I_3}(B_3) &= \emptyset \subseteq \text{dom}_{I_3}(B_2) = \emptyset, \\ \text{dom}_{I_4}(B_1) \cap \text{dom}_{I_4}(B_3) &= \emptyset \subseteq \text{dom}_{I_4}(B_2) = \emptyset. \end{aligned}$$

Finally, we may compose these pos τ -blocks with $\text{car}(C')$ to obtain the resulting set of neutral features.

In principle, it could happen that by removing some redundant pos τ -blocks from \mathcal{F} , another redundant pos τ -block could become irredundant. While it indeed happens for non-strictly redundant blocks (if we have only two blocks with equal domains and we remove one of them, the second one will become irredundant), it does not happen if we filter the non-strictly redundant blocks before we start the process of discarding the strictly redundant ones. This is expressed more formally by the next proposition.

Proposition 13. *Let $\mathcal{F} = \{B_i\}_{i=1}^n$ be a set of pos τ -blocks with equal types of input arguments such that no two pos τ -blocks in the set \mathcal{F} have equal domains for all examples. Let \mathcal{F}' be a set of pos τ -blocks obtained from \mathcal{F} by repeatedly removing redundant pos τ -blocks. Set \mathcal{F}' is unique.*

Algorithm 3 RELF: Given a template and a set of examples, RELF computes the propositionalized table.

- 1: **Input:** template τ , examples E ;
 - 2: $\text{PosBlocks} \leftarrow \{\}$
 - 3: $\text{OrderedAtoms} \leftarrow$ topologically ordered (w.r.t. \prec from Def. 17) predicate definitions computed from τ
 - 4: **for** $\forall \text{Atom} \in \text{OrderedAtoms}$ **do**
 - 5: $\text{NewPosBlocks} \leftarrow \text{Combine}(\text{Atom}, \text{PosBlocks}, E)$ // See procedure Combine in Algorithm 4
 - 6: Filter pos τ -blocks with equal domains for all examples from (keep the short ones and if they have equal sizes, select them arbitrarily) NewPosBlocks
 - 7: Filter redundant pos τ -blocks from NewPosBlocks
 - 8: Add NewPosBlocks to PosBlocks
 - 9: **end for**
 - 10: Save all correct features from PosBlocks
-

5.8 FEATURE CONSTRUCTION ALGORITHM RELF

In this section, we combine the ideas from the previous sections and present a description of the algorithm RELF⁴ (Algorithm 3). The described algorithm constructs features using the graft operator and it filters reducible and redundant features using the rules presented in previous sections. The next proposition asserts the completeness of the algorithm. In the subsequent comments, we address some details of the algorithm and also provide the intuition as to why the completeness property holds.

Proposition 14. *Let $\tau = (\gamma, \mu)$ be a template and let $E = E^+ \cup E^-$ be the set of examples. Further, let \mathcal{F}_τ be the set of all non-H-reducible τ -features and let $\mathcal{F}_{\text{RELF}}$ be the set of conjunction of atoms constructed by RELF. Then $\mathcal{F}_{\text{RELF}} \subseteq \mathcal{F}_\tau$ and for every τ -feature $F_\tau \in \mathcal{F}_\tau$, which is not strictly redundant, there is a feature F_{RELF} such that $\text{ext}_E(F_{\text{RELF}}) = \text{ext}_E(F_\tau)$.*

Let $\tau = (\gamma, \mu)$ be a template. Let us tackle a simplified task first and let us construct all non-H-reducible τ -features. The algorithm takes pos blocks and grafts them with atoms and other pos blocks in order to produce bigger blocks and continues with this process until it produces the whole set of features. We have not explained yet how the order of atoms from τ should be determined. It is obvious that we could always start with the atoms that are declared in τ without output arguments (e.g. $\text{box}(+l)$ or $\text{tri}(+l)$). Next, we would like to graft these one-atom blocks with other atoms from τ . At any point of the feature construction process, only the atoms with *types* of output arguments equal to the types of positive variables of the already generated blocks are usable. The question is whether we can expect that such atoms always exist. The answer is that if the set of features is non-empty and if it has not been constructed yet then such atoms must always exist, which is a consequence of the partial order \prec on constants (i.e. *types*) given in Definition 17.

Example 22. Consider the template

$$\tau \approx a(-a), a(+b), a(+c), \text{bond}(+a, -b, -t), \text{bond}(+b, -c, -t), \text{single}(+t), \text{double}(+t).$$

RELF starts by building blocks corresponding to $a(+b)$, $a(+c)$, $\text{single}(+t)$ or $\text{double}(+t)$. Assume it has already built blocks corresponding to all these four atoms. Next, the only pos-

⁴ The original implementation of RelF used for the experiments in the paper [62] is publicly available at <http://ida.felk.cvut.cz/RELF>. A newer version is available as a part of the TreeLiker suite of relational learning algorithms, described in Appendix, at <http://ida.felk.cvut.cz/treeliker/>.

Algorithm 4 Combine: Given a set of already generated pos τ -blocks, an atom $A \in \gamma$ and a set of examples, this procedure generates τ -features or pos τ -blocks with their roots built upon the predicate symbol of A .

```

1: Input: Atom  $A \in \gamma$ , Set of pos  $\tau$ -blocks PosBlocks, Set of examples Examples =  $E^+ \cup E^-$ ;
2: Generated0  $\leftarrow$  {a}, where a is an atom built upon the predicate symbol of A
3: for  $i = 1, \dots, \text{arity}(A)$  do
4:   if  $i$ -th argument of A is an output then
5:     Generated $i$   $\leftarrow$  {}
6:     /* Procedure BuildCombinations(PosBlocks, Type, Examples) constructs the set of
7:     all legal combinations of pos blocks with the type of positive variables equal to Type */
8:     Combinations  $\leftarrow$  BuildCombinations(PosBlocks, Type, Examples), where Type is
9:     the term appearing as  $i$ -th argument of A
10:    for  $\forall B^- \in \text{Generated}_{i-1}$  do
11:      Let V be the  $i$ -th argument of a
12:      for  $\forall \mathcal{S} \in \text{Combinations}$  such that  $\exists I \in E^+ : \text{dom}_I(B^- \oplus_V \mathcal{S}) \neq \emptyset$  do
13:        /*  $\mathcal{S}$  is a set of pos blocks */
14:        F  $\leftarrow$   $B^- \oplus_V \mathcal{S}$ 
15:        Generated $i$   $\leftarrow$  Generated $i$   $\cup$  {F}
16:      end for
17:    end for
18:    Filter  $E^+$ -redundant pos  $\tau$ -blocks from Generated $i$ 
19:  else
20:    Generated $i$   $\leftarrow$  Generated $i-1$ 
21:  end if
22: end for
23: return Generated $i$ 

```

sibility is to take the atom $\text{bond}(+b, -c, -t)$ because if $\text{bond}(+a, -b, -t)$ were taken before $\text{bond}(+b, -c, -t)$, it would not be possible to construct all features. For example

$$a(A), \text{bond}(A, B, T1), \text{bond}(B, C, T2), a(C), \text{single}(T1), \text{double}(T2)$$

could not be constructed as can be easily seen. So it must take $\text{bond}(+b, -c, -t)$ and only after that it can take $\text{bond}(+a, -b, -t)$. In the end it can build features by grafting the already generated blocks with $a(-a)$.

When we have a procedure capable of constructing the whole set of non-H-reducible features, we will be able to enrich it by detection of redundant blocks. Due to monotonicity of redundancy we will still be able to obtain one feature for each set of irredundant features with equal extensions.

Consider now the step in which blocks are composed to yield bigger blocks. It is quite straightforward to construct the set of all pos blocks with an atom a as root when we already have the set of all *legal* combinations of the pos τ -blocks that were already generated (cf. Algorithm 4). Legal combinations of pos τ -blocks are those combinations where all pos blocks B_i have equal *type* of the variable $p(B_i)$ (if τ is ambiguous and if we denote by $\hat{p}(B_i)$ the set of all possible *types* of the positive variable of B_i then we require the intersection $\cap_i \hat{p}(B_i)$ to be non-empty) and for $i \neq j$, we require $B_i \not\leq_H B_j$ (i.e. we require that no combination gives rise to H-reducible blocks). Finding the legal combinations is a rather straightforward task, which can be also combined with redundancy filtering.

5.9 FEATURE CONSTRUCTION ALGORITHM HIFI

The feature construction algorithm RELF needs its input in the form of labelled learning examples because filtering based on the notion of strict redundancy would not work without class-labels. On the other hand, class-labels would not be necessary if we wanted to use only filtering based on non-strict redundancy. Recall that two features are mutually non-strictly redundant if they have equal domains w.r.t. all training examples. The algorithm presented in this section, called HiFi⁵, is very similar to RELF. Unlike RELF, it uses only filtering based on non-strict redundancy and its language bias allows setting maximum size of features. HiFi does not require labelled learning examples therefore it can be used also for unsupervised learning tasks, e.g. for clustering.

The algorithm HiFi is almost identical to RelF. There are just two differences described in the next paragraphs. First, where RelF performs filtering of redundant pos blocks, HiFi filters only non-strictly redundant pos blocks, i.e. it removes all but one pos block for every set of pos blocks with the same domains. Second, HiFi is also able to detect that a pos block or a combination of pos blocks cannot be extended to a feature that would have smaller or equal number of literals than a given size limit. This is used in HiFi to filter candidate pos blocks when they are created by grafting and candidate combinations of pos blocks when they are created (which corresponds to line 7 in Algorithm 4 - algorithm Combine). How HiFi actually checks which blocks can be extended to features satisfying given maximum-size constraints is explained next.

Despite the fact that checking whether a pos block can be extended to feature satisfying the given maximum-size limit may seem as a trivial problem, it is not trivial, and, in fact, it is NP-hard if we allow slightly more general type of (non-treelike) features. In Section 5.14, we show formally that for general features, bounding features' size is one of the factors that makes it NP-hard even to decide whether at least one feature exists. This problem is no longer NP-hard if we constrain ourselves to treelike features as noted in [130]. However, it is important to stress that the obvious approach, which would discard pos blocks with size greater than the limit, would not be very efficient even for treelike features because it could often leave many pos blocks unfiltered. Here, we describe a simple and efficient method for checking whether a pos τ -block may be extended to a treelike τ -feature with size not greater than some limit.

In what follows, we will speak of *declared predicates from template* τ . If $\tau = (\gamma, \mu)$ is a template, the term *declared predicate* will refer to a ground atom $l \in \gamma$ together with the respective argument places $\mu_l \subseteq \mu$, e.g. `hasLoad(+c, -l)` will be a declared predicate. Let m denote the number of predicates declared in a template τ and let a denote maximum arity of the predicates. We now show how, for all types t , we can find sizes of the smallest pos blocks B_{\min} such that B_{\min} has t as input type of its root in time $O(m^2 + m \cdot a)$. We start by finding a topological order of the graph induced by the given template. As this graph surely has less than m^2 edges, it is possible to find its topological ordering in time $O(m^2)$. When the topological ordering is found, we can take the declared predicates starting with input-only predicates and for every such predicate, we can compute size of the corresponding smallest pos block. To accomplish this, we sum sizes of smallest pos blocks whose roots have input types equal to output types of the respective processed predicate declaration. These sizes must have been already computed due to the topological order assumed. Since this is done for all m declared predicates, it follows that computing sizes of the desired smallest pos blocks takes time $O(m^2 + m \cdot a)$.

Since pos blocks are composed into bigger pos blocks (and eventually features) from smaller blocks, what we really need to be able to determine efficiently is the size of the smallest feature containing a given pos block. Let m denote the number of predicates declared in a template τ and let a denote maximum arity of the predicates. Then, for all predicates p , we can find sizes of the smallest treelike features F_{\min}^p containing p in time $O(m^2 + m \cdot a)$. Using the result from the previous paragraph, we can translate this problem to a problem of finding shortest paths in an acyclic graph G as follows. Let V be a set of vertices of G and let each vertex correspond to a

⁵ HiFi stands for **h**ierarchical feature construction which is a term that we originally used in [69] for what is termed *block-wise* in this thesis.

predicate declared in τ . Let there be an oriented edge e between predicates p_1 and p_2 if and only if type t of the input argument of p_2 is equal to the type of some of the output arguments of p_1 . Further, let the weight of each edge be defined according to Eq. 5, where $\text{MinSize}(t)$ refers to minimum size of a feature with input-type of its root equal to t .

$$w(p_2, p_1) = 1 - \text{MinSize}(\text{Input}(p_2)) + \sum_{t \in \text{Outputs}(p_1)} \text{MinSize}(t) \quad (5)$$

The size of the smallest feature, which contains p , is then given as the length of the shortest path from p to the only vertex with no inputs plus the size of the smallest pos block having p as its root. The number of edges of G is bounded by m^2 and the graph is acyclic, therefore we can compute all the shortest paths in time $O(m^2)$. Thus, we see that it is possible to compute smallest sizes of features containing some declared predicate in time $O(m^2 + m \cdot a)$.

Thus, we are able to decide whether a pos block B can be extended to a correct treelike feature with size less than some n . This is because the minimum possible size of a treelike feature containing B can be computed as the size of the smallest feature containing root of B , from which we subtract the size of the smallest pos block having the same root as B and to which we add the size of B .

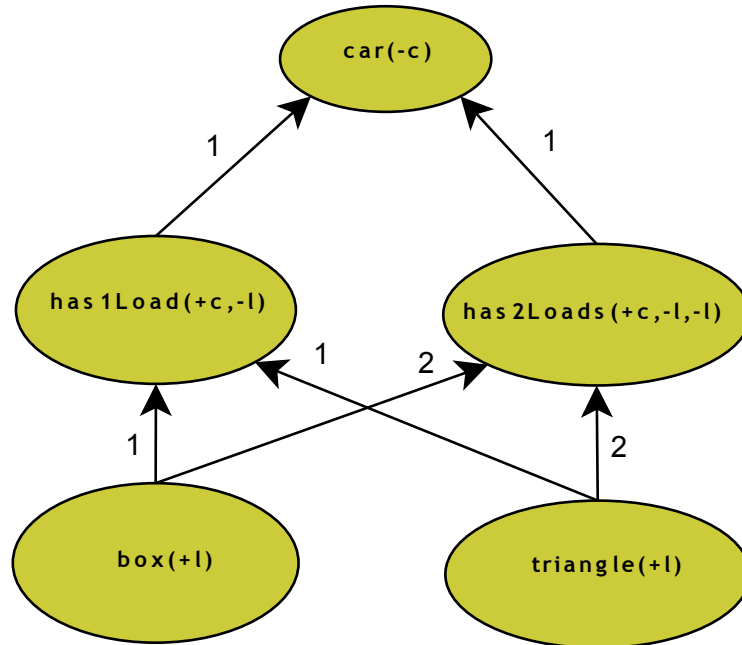


Figure 6: An example graph corresponding to template used in Example 23. Edge labels are computed from Eq. refeq:eq1.

Example 23. Let us compute sizes of the smallest features, which contain some declared predicate, which are defined by the following template.

$$\leftarrow \text{car}(-c), \text{has1Load}(+c, -l), \text{has2Loads}(+c, -l, -l), \\ \text{box}(+l), \text{triangle}(+l)$$

First we find the topological ordering on types, which is (L, C) . Then we can compute sizes of smallest pos blocks with given roots:

$$\text{box}(+l) \rightarrow 1, \text{triangle}(+l) \rightarrow 1, \text{has1Load}(+c, -l) \rightarrow 2, \\ \text{has2Loads}(+c, -l, -l) \rightarrow 3, \text{car}(-c) \rightarrow 3$$

Now, we need to compute sizes of treelike features which contain given declared predicates. In order to do so, we build the corresponding graph (Fig. 6) and compute shortest paths. It follows that e.g. the smallest feature containing predicate $\text{has2Loads}(+c, -l, -l)$ has size $1 + 3 = 4$ (1 is the length of the path from has2Loads to car and 3 is the size of the smallest subfeature having has2LoTad as root) and e.g. the smallest feature containing predicate $\text{box}(+l)$ has size $2 + 1 = 3$.

Being able to decide whether a pos block B can be extended to a correct feature F with size smaller than n is an important factor contributing to efficiency of HiFi. In Section 5.14.2, we show that HiFi can construct the complete set of valid τ -features for a given template τ in output-polynomial time (Proposition 16).

5.10 EXPERIMENTS

In this section we evaluate the speed of RELF and the accuracy of classifiers constructed with RELF's features. Our first intention is to compare the runtimes of feature construction conducted by RELF to those achieved by the existing propositionalization system RSD. In [59] RSD has been shown to be competitive w.r.t. predictive accuracy with propositionalization system SINUS [59] and RELAGGS [58] in typical ILP tasks such as predicting mutagenicity or learning legal positions of chess-end-games, therefore we chose RSD for comparison with RELF. Our second goal is to assess the classification accuracies obtained with RELF's features in nine relational classification benchmarks. Nine different methods for classification combined from propositionalization and learning algorithms (RELF, RSD, Support Vector Machines, Logistic Regression, One Rule, kFoil, nFoil and a RELF's variation entitled nRELF) are tested in this study. Lastly, we aim at tracing the effects of reducibility and redundancy based pruning on RELF's performance.

In the experiments described in this section we use l_2 -regularized logistic regression from LIBLINEAR package [30] and support vector machines [126] with RBF kernel from WEKA [133] for all datasets except for the NCI dataset where we use linear SVM from LIBLINEAR package which is a faster alternative to the RBF kernel SVM. We follow suggestions given in [107] to obtain an unbiased estimate of quality of learned classifiers. All accuracies presented in this section are estimates obtained by 10-fold cross-validation except the accuracies for the NCI 786 dataset for which a train-test split was used. For each fold, the parameters (maximum depth of features for RELF, maximum feature length for RSD and the cost parameter for SVM and logistic regression) are optimized by 3-fold cross-validation on the remaining nine training folds. We perform experiments both with RSD having the same feature declaration bias as RELF and with RSD allowing cyclic features.

5.10.1 Datasets

We performed experiments with four molecular datasets and two datasets related to engineering problems. The first experiment was done with the regression-friendly part of the well-known Mutagenesis dataset [43], which consists of 188 organic molecules marked according to their mutagenicity. The next set of experiments was done with data from the Predictive Toxicology Challenge [46] which consists of more than three hundreds of organic molecules marked according to their carcinogenicity on male and female mice and rats. We also performed experiments with a dataset that we created by merging the Carcinogenesis dataset [113], the PTC dataset and the Mutagenesis dataset. The task in this slightly artificial learning problem was to distinguish the generally *toxic* compounds from the remaining *control* compounds. The largest of the four molecular datasets that we use in this chapter is the NCI 786 [115] dataset which contains 3506 molecules labeled according to their ability to inhibit growth of renal tumors. In all of these four experiments, we used only atom-bond descriptions and only the crude resolution with atom types like *carbon*, *hydrogen* and not the finer resolution with atom types like *aliphatic hydrogen* or *aromatic hydrogen*.

Dataset	Classes	Examples	Facts	Acc. of Majority Class
Mutagenesis	2	188	21024	66.5 %
CAD	2	96	16674	57.2 %
Mesh	13	278	2387	26.4 %
CPM	2	830	67309	52.9 %
P-FM	2	349	27762	59.0 %
P-MM	2	336	25482	61.6 %
P-FR	2	351	27762	65.5 %
P-MR	2	344	26986	55.8 %
NCI 786	2	3506	254380	52.3 %

Table 3: Properties of Mutagenesis dataset (**Muta**), Predictive Toxicology Challenge dataset (**PTC**), CAD dataset (**CAD**), Mesh dataset (**Mesh**) and Carcinogenesis + PTC + Mutagenesis dataset (**CPM**)

Next, we performed experiments in a domain describing CAD documents (product structures) [139]. The CAD dataset is interesting in that relatively long features are needed to obtain reasonable classification accuracy. An example of a feature from this dataset is

$$F = \text{cadFile}(A) \wedge \text{hasCADEntity}(A, B) \wedge \text{hasBody}(B, C) \wedge \text{hasFeature}(C, D) \wedge$$

$$\wedge \text{hasSecondLimit}(D, E) \wedge \text{limitType}(E, \text{offset}) \wedge \text{next}(D, F) \wedge \dots 18 \text{ more atoms.}$$

Finally, we performed experiments with the well-known Mesh dataset [26]. The task in this problem is to predict, for each edge of a geometrical model, the best number of finite elements that should be placed on the edge in order to introduce small approximation errors to finite element modeling while keeping the complexity of the model as low as possible. The examples in the Mesh dataset are not in the form of isolated interpretations, but they form several interconnected *networks* corresponding to particular finite element models. The current implementation of RELF copes with this problem by reading the *networks* and converting them to single interpretations according to a given template τ . The conversion process accepts a *network* \mathcal{N} together with classification of the nodes and a template $\tau = (\gamma, \mu)$ where the declared predicates with no input arguments are supposed to refer to nodes in the network. An interpretation corresponding to a node $n \in \mathcal{N}$ is then $I_n = \cup_{A, \theta} A\theta$ where A ranges over all connected subsets of γ which contain a declared predicate with no input arguments and θ ranges over all substitutions such that $A\theta \subseteq \mathcal{N}$. The interpretations can be computed efficiently e.g. using breadth-first-search, essentially resembling the way bottom clauses are computed in Progol [86].

5.10.2 Feature Construction

Here we present feature construction runtimes of RELF and RSD on the six datasets. Since RELF does not restrict size of features, we performed experiments using templates with varying complexity (depth). For RSD we used templates corresponding to the most complex templates used by RELF but we limited their maximum size. Figure 7 displays runtimes of RELF and RSD. For the molecular datasets, we report the number of *bond*-atoms in the longest features since this is a more intuitive measure of their complexity than the number of logical atoms. For the remaining two datasets, we report numbers of logical atoms in the longest features.

In all of the experiments, RELF was able to construct significantly larger features than RSD. The longest features discovered by RELF for the molecular datasets contained more than twenty *bond* atoms. This contrasts with results obtained by RSD, which was able to find features with at most four *bond* atoms. Only, in the experiment with the NCI 786 dataset using the most complex template, we also needed to set minimum frequency of RELF's features to 1%. A consequence

of this was that even if we had not restricted RSD's bias to treelike features, RSD did not find any cyclic features (only, it took longer to generate the treelike ones). For the remaining two datasets, CAD and Mesh, RELF was again able to construct far larger features than RSD but in the case of the CAD data, RSD was also able to construct cyclic features. However, the predictive accuracy for the cyclic features was lower than for the acyclic case. We also measured memory consumption of RELF. Since our current implementation of RELF exploits properties of Java garbage collection mechanism, maximum memory consumption measured with a loose memory limit can be higher than memory consumption measured with some lower memory limit. Therefore we used RELF with maximum memory set to 128MB, 256MB etc. and we report the minimum memory that enabled RELF to finish its execution. RELF needed 256MB for all four PTC datasets and for Mutagenesis. It needed 768MB for the CPM dataset. On the CAD dataset, RELF was able to run with maximum memory set to just 64MB and with 128MB on the Mesh dataset. For the NCI 786 dataset, RELF needed 4GB of memory on a 64-bit machine.

We also performed several experiments in order to assess the effect of the pruning methods implemented in RELF on its performance. First, we disabled filtering of redundant features. A result of this was that RELF was only able to construct features corresponding to the templates with lowest complexity (cf. Fig. 7) on molecular datasets. With turned off filtering of redundant features, RELF was unable to construct features on the CAD dataset. This clearly illustrates that filtering of redundant features is an essential part of RELF. To evaluate the effect of filtering of H-reducible blocks, we turned off explicit tests for H-subsumption in the phase where combinations are built. This resulted in about 40% slow-down on CAD dataset and negligible slow-downs on the rest of the datasets. When judging the impact of H-subsumption filtering integrated in RelF, it is important to appreciate that this method serves not only to perform these explicit tests but also to make RELF able to consider only combinations of blocks without repetitions.

The systems kFOIL and nFOIL do not resort to exhaustive search, instead, they are based on beam search. This means that by setting beam size and maximum size of searched clauses sufficiently low, these algorithms can find reasonably accurate classifiers in a short time. In order to test kFOIL and nFOIL, we set these parameters so that their running times would be in the same orders as the running time of RELF and then compare the obtained predictive accuracies. nFOIL was used with maximum clause length set to 20 for all datasets and with beam size set to 100 for Mutagenesis, 30 for CAD and PTC, 10 for Mesh and NCI 786 dataset and 20 for CPM. Parameter settings for kFOIL were as follows. Beam size was set to 100 for CAD, to 200 for Mutagenesis, to 10 for PTC and CPM and to 7 for NCI 786. Maximum clause length was set to 30 for Mutagenesis and CAD datasets, to 5 for P-MR P-FM, P-MM, P-FR, CPM and NCI 786. It is interesting to note that kFOIL's runtime varied significantly even for very similar datasets. For example, while it took kFOIL just several minutes to finish on the P-FM dataset, it needed tens of hours to finish on the P-MR dataset.

5.10.3 Classification

Here we present classification accuracies obtained using RELF's or RSD's output in conjunction with propositional learners and accuracies obtained by kFOIL and nFOIL. Table 4 displays predictive 10-fold cross-validated accuracies obtained using one best rule, support vector machines and l_2 -regularized logistic regression acting on features constructed by RELF and RSD. An exception was the NCI 786 dataset for which accuracy was assessed using a train-test split. RELF obtained highest accuracy on six out of nine datasets. RELF in conjunction with SVM or logistic regression also obtained higher predictive accuracy than nFOIL and kFOIL.

The higher accuracies achieved by RELF could have been attributed to the fact that RELF creates classifiers using several thousands of features whereas kFOIL and nFOIL try to build classifiers using only a small set of informative features. Thus, in the experiments with SVM and logistic regression, we were essentially trading interpretability for predictive accuracy. In order to assess whether RELF's ability to construct large features is beneficial also for the task

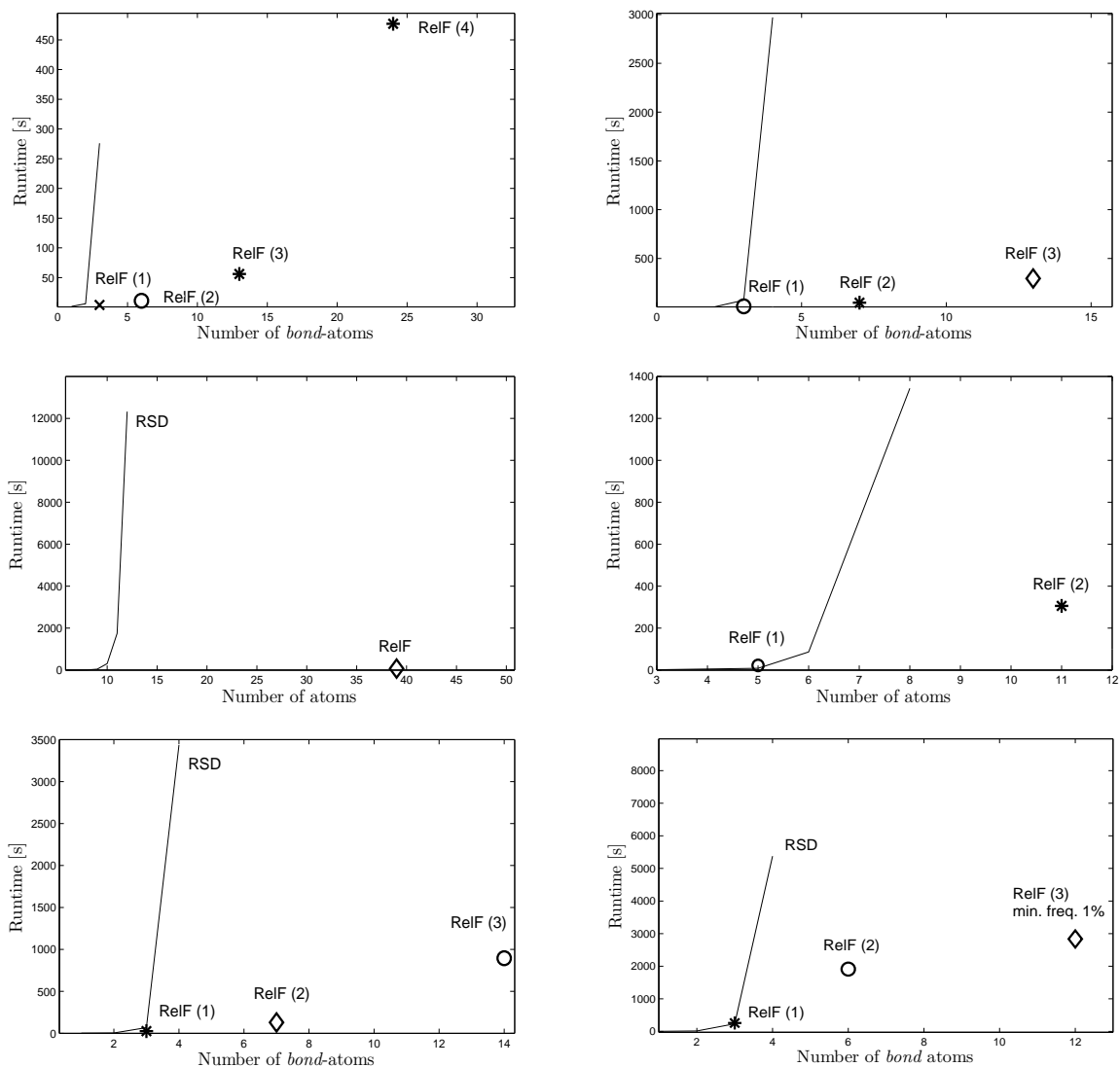


Figure 7: Runtimes of RELF and RSD on Mutagenesis (top left), PTC-MR (top right), CAD (middle left) Mesh (middle right), CPM (bottom left) and NCI 786 (bottom right). The syntactical bias of RSD allows to specify the maximum allowed length of features. RELF only allows to specify a template and, implicitly, also the maximum depth of the features. Therefore RSD is used with template corresponding to the template of RELF with the greatest complexity and limited maximum length and RELF is used with several templates with increasing complexity.

	RelF			RSD		
	SVM	LR	OneR	SVM	LR	OneR
Muta	87.4 ± 6.9	81.1 ± 8.9	70.3 ± 4.8	79.9 ± 4.6	82.6 ± 6.8	72.3 ± 4.8
CAD	96.8 ± 5.2	96.8 ± 5.2	85.5 ± 8.6	94.9 ± 7.2	95.1 ± 9.7	81.4 ± 9.3
Mesh	62.6 ± 6.9	60.7 ± 10.3	40.9 ± 5.4	61.4 ± 7.3	58.9 ± 6.2	40.7 ± 4.2
CPM	73.6 ± 4.5	71.2 ± 4.5	56.9 ± 5.2	71.3 ± 3.7	68.7 ± 3.9	56.5 ± 1.7
P-FM	62.2 ± 5.7	57.9 ± 6.3	64.4 ± 6.0	59.3 ± 6.9	58.7 ± 5.3	64.5 ± 3.6
P-MM	59.9 ± 8.3	63.4 ± 10.5	64.9 ± 5.8	62.2 ± 4.9	59.1 ± 6.8	61.3 ± 9.6
P-FR	66.4 ± 3.0	65.5 ± 4.5	68.6 ± 3.4	65.3 ± 6.0	67.3 ± 6.0	68.7 ± 4.1
P-MR	62.0 ± 6.9	66.0 ± 5.6	57.8 ± 5.3	65.1 ± 6.1	69.2 ± 6.0	56.2 ± 5.8
N 786	68.3	69.6	56.0	69.3	69.3	55.2

Table 4: Accuracies of RELF and RSD on Mutagenesis dataset (**Muta**), Predictive Toxicology Challenge datasets (**P-FM**, **P-MM**, **P-FR**, **P-MR**), CAD dataset (**CAD**), Mesh dataset (**Mesh**), Carcinogenesis + PTC + Mutagenesis dataset (**CPM**) and NCI 786 dataset (**N 786**)

	nRELf	nFOIL	kFOIL
Muta	80.6 ± 9.8 (9.7)	76.6 ± 9.6 (4.6)	76.0 ± 5.6 (7.1)
CAD	94.1 ± 6.7 (3)	92.7 ± 6.9 (3.6)	89.8 ± 10.6 (2)
Mesh	59.4 ± 4.2 (16.2)	57.9 ± 11.6 (18.5)	n.a.
CPM	65.1 ± 2.8 (65.3)	58.3 ± 5.8 (9)	62.2 ± 4.7 (16.9)
P-FM	61.5 ± 9.3 (22.5)	60.2 ± 8.7 (5)	57.3 ± 5.6 (3.1)
P-MM	61.0 ± 6.9 (26.6)	63.1 ± 8.8 (6.8)	62.2 ± 7.4 (4.6)
P-FR	70.7 ± 5.6 (21.9)	67.0 ± 6.5 (9.1)	63.4 ± 8.3 (6.7)
P-MR	53.9 ± 10.0 (28.6)	57.3 ± 7.1 (6.7)	59.3 ± 7.0 (7.2)
N 786	64.4 (41)	63.7 (16)	63.1 (5)

Table 5: Accuracies of RELF, kFOIL and nFOIL on Mutagenesis dataset (**Muta**), Predictive Toxicology Challenge datasets (**P-FM**, **P-MM**, **P-FR**, **P-MR**), CAD dataset (**CAD**), Mesh dataset (**Mesh**), Carcinogenesis + PTC + Mutagenesis dataset (**CPM**) and NCI 786 dataset (**N 786**). The numbers in parentheses denote average number of features in the respective classifiers.

of finding interpretable classifiers we performed an additional experiment. We followed the approach introduced in SAYU [21] with features generated by RELF. We started with an empty set of features and we greedily added features that improved accuracy of a naive Bayes classifier⁶. This method, marked as nRELf in Table 5, obtained higher predictive accuracy than kFOIL and nFOIL on seven out of nine datasets which shows that RELF’s ability to construct large features is beneficial also for the task of finding small interpretable classifiers.

5.11 RELATED WORK

The primary aim of the algorithm RELF presented in this chapter is to enable exhaustive generation of large numbers of features in a limited subset of first order logic. Several pattern mining algorithms have been proposed for this task or for more or less similar tasks. A well-known al-

⁶ This was implemented using WEKA [133] using its *AttributeSelectedClassifier* with feature selection method *Classifier-SubsetEval* and with search procedure *GreedyStepwise*. Since this turned out to be very time-consuming, we used only the features from the NCI 786 dataset that had frequency higher than 1%.

gorithm working in the logical setting is the frequent pattern miner WARMR [25], which greatly exploits monotonicity of frequency to prune uninteresting patterns. Monotonicity of frequency can be easily exploited also in our algorithm by discarding infrequent pos blocks⁷. If a minimum frequency threshold is set in RELF, however, one has to refine the statement about RELF's completeness as only the irredundant features with frequency above the specified threshold are guaranteed to be found. Another algorithm for pattern discovery is the RSD algorithm [131]. RSD does not prune patterns using the minimum frequency constraint as WARMR does, instead, RSD relies on its expressive user-definable syntactical bias, which is also somewhat similar to WARMR's warmode declarations. RSD's language bias is also used by RELF which puts further requirements on syntax of correct features ensuring their treelike structure. A common principle of RSD and WARMR is the level-wise approach to feature construction, which means that features are built by adding one logical atom at time. Unlike the block-wise approach presented in this chapter, the level-wise approach to feature construction cannot be easily combined with filtering of reducible and redundant features, which we have already discussed in the respective sections about redundancy and relevancy.

A recent line of research, represented by algorithms nFOIL [74], kFOIL [73] and SAYU [21], tries to refrain from explicitly constructing the set of all *interesting* features (frequent, non-redundant etc.) by constructing only features that improve classification accuracy or some related scoring criterion when combined with a propositional learner such as Naive Bayes or SVM. Both nFOIL and kFOIL used in experiments in this chapter are based on FOIL's [99] search, although, in principle, any strategy for hypothesis search could be used instead of FOIL. A potential replacement for FOIL could be e.g. Progol [86] or even some procedure searching over features generated by a propositionalization algorithm. An advantage of systems like nFOIL and kFOIL is that they can produce relatively accurate models within reasonably short runtimes. On the other hand, in our experiments, giving these algorithms more time did not increase their predictive accuracy sufficiently to enable them to outperform classifiers constructed from the whole set of RELF's features.

Frequent graph mining algorithms are also related to RELF. They are very well suited for molecular databases, however, they have limitations in other domains. For example, the currently fastest graph mining algorithm Gaston [93] is able to mine frequent molecular structures from larger databases than RELF, but it cannot easily handle oriented graphs or even hypergraphs [134]. The covering relation adopted by graph mining algorithms, which corresponds to subgraph isomorphism, differs significantly from the one used by RELF and other related algorithms such as nFOIL or kFOIL, whose covering relation corresponds to hypergraph homomorphism. As a consequence, number of frequent patterns in the respective frameworks may differ significantly as, for example, some patterns are expressible only with homomorphism. Conversely, there are tree patterns that can be represented with the graph mining approach but cannot be represented by treelike features with their covering relation. Another notable system geared towards discovery of patterns in molecular databases is MolFea [57], which restricts the patterns to linear molecular fragments.

Recently, there has been a growing interest in removing various forms of *redundancy* from frequent pattern sets. The form of *redundancy* exploited by RELF was introduced in [77] where it was defined for propositional data and for constrained Horn clauses, which are equivalent to propositional representation in their expressive power. RELF extends this framework to treelike features by establishing monotonicity of *redundancy* w.r.t. grafting. There are also other forms of *redundancy* considered in literature. For example, a system called Krimp was introduced in [56, 125] which searches for a representative set of features that would allow for a lossless compression of a given set of examples, i.e. it relies on the minimum description length principle. Mining of closed frequent patterns is another approach that minimizes the number of discovered patterns by considering only representative candidates from certain equivalence classes. Finally, in [12], it has been shown that simple patterns (e.g. linear fragments) work well when employed

⁷ It is possible to set minimum frequency in the current implementation of RELF.

in classification of chemical compounds, which is directly related to our work as we also use only treelike patterns.

5.12 CONCLUSIONS

In this chapter, we have introduced RELF, an algorithm for construction of treelike relational features. We have shown that block-wise construction of treelike features enables RELF to remove H-reducible and redundant pos blocks. In experiments, we have shown that RELF is able to construct *relevant* features with sizes far beyond the reach of state-of-the-art propositionalization systems. Of importance, we have also shown that, for nine relational learning benchmarks, restriction to treelike features was not detrimental with respect to predictive accuracy. In fact, the results obtained with treelike features were, in most cases, higher than predictive accuracies of state-of-the-art systems nFOIL and kFOIL. Importantly, the features constructed by the other tested algorithms were often also treelike. Although there are definitely datasets where cyclic features could provide better predictive accuracies, one can always merge results from different propositionalization algorithms and feed the propositional learners with such merged propositionalized tables. An algorithm for construction of a limited class of features such as RELF would be useful also in such cases.

5.13 PROOFS

For sakes of the following proofs we first need to define a few concepts we did not need in the main text.

Definition 28 (pos-neg τ_θ -block). *Let B be a τ_θ -conjunction. If there is exactly one τ_θ -positive variable ($p(B)$) and exactly one τ_θ -negative variable ($n(B)$), we say that B is a pos-neg τ_θ -block.*

Definition 29 (Descendant, Induced pos τ -block). *Let F be a $(\gamma, \mu)_\theta$ -feature or a pos τ -block and let $a \in F$ be an atom. We say that an atom $b \in F$ is a descendant of a (denoted by $a \prec_d b$) if there is a sequence of atoms a_1, \dots, a_n such that $a_1 = a$, $a_n = b$ and every two consecutive atoms a_i, a_{i+1} share a variable which has an output in $a_i\theta$ and an input in $a_{i+1}\theta$. Then $\text{Induced}_F(a) = \{a\} \cup \{b \mid a \prec_d b\}$ is called a pos τ -block induced in F by a .*

Definition 30 (Parent, Child). *Let F be a $(\gamma, \mu)_\theta$ -feature or a pos τ -block. If two atoms $a, b \in F$ share a variable V , which has an output in $a\theta$ and an input in $b\theta$, we call a parent of b (denoted by $P_F(a)$) and b child of a (denoted by $C_F(a)$).*

Notation	Meaning	Note
$b \in C_F(a)$	b is a child of a	a and b are atoms
$b = P_F(a)$	b is a parent of a	a and b are atoms
$a \prec_d b$	a is a descendant of b	a and b are atoms
$B = \text{Induced}_F(a)$	B is an induced pos block of a in F	a is an atom, B, F are blocks/features

Table 6: Summary of notation introduced in Definitions 29, 30

Lemma 1. *Let F' be the result of the reduction from Def. 38 applied on a τ -feature F . Then the number of atoms in F' is the same as the number of variables in F' . Furthermore, each atom in F' contains an output and exactly one input.*

Proof. Denote $A(V)$ the number of atoms (variables) in F' . Every variable in F' must have at least two occurrences (otherwise it would have been removed by rule 2) and one of them must

be input as each variable has at most one output in F (due to the neutrality requirement) and thus also in F' . Each atom in F , and thus in F' , contains at most one input due to condition (i) in Def. 17, so $A \geq V$. Furthermore, each atom in F' contains more than one term (otherwise it would have been removed by rule 1) and, as we have seen already, contains at most one input. So each atom in F' contains an output. Again realize there is at most one output per variable in F' , therefore $V \geq A$. We have seen that $A \geq V$, so indeed $A = V$ as the lemma says. We have also seen that each atom contains an output which was to be shown. Lastly, we have seen that each of the $A = V$ atoms contains at most one input and each of the $A = V$ variables must have one input in F' so each atom has exactly one input. \square

Proposition 8 *Every τ -feature F is treelike.*

Proof. Since F is a τ -feature, $\tau = (\gamma, \mu)$, there must be a substitution θ such that $F\theta \subseteq \gamma$. By Def. 17 there is a partial order \prec on $\{v_i\theta \mid v_i \in \text{vars}(F)\}$ such that $v_j\theta \prec v_k\theta$ whenever v_j (v_k) has an input (output) in an atom $a \in F$. Let F' be the result of the reduction from Def. 38 applied on F . Denote \prec' the suborder of \prec on $\text{vars}(F') \subseteq \text{vars}(F)$. Since \prec' is also a partial order, we can sort variables in F' topologically as $v_1, v_2 \dots v_n$ ($n = |\text{vars}(F')|$) such that

$$v_j\theta \prec' v_k\theta \Rightarrow j < k \quad (6)$$

According to Lemma 1 there are exactly n atoms in F' each containing exactly one input. Assume for contradiction that F is not treelike and thus $n > 0$. We sort atoms in F' by their single inputs, i.e. atom a_i has v_i as input. Atom a_n that has v_n as input must, by Lemma 1, have another variable v_m ($1 \leq m \leq n$) as output. Since v_n (v_m) has an input (output) in atom a_n , it must be that $v_n \prec' v_m$. This however contradicts implication 6 above as $m \leq n$. Therefore F is treelike. \square

Lemma 2. *Let A, B be $(\gamma, \mu)_\theta$ -features or pos (γ, μ) -blocks, let ϑ be an H-substitution such that $A\vartheta \subseteq B$ (i.e. $A \preceq_H B$). If $b \notin A\vartheta$ and $b' \in \text{Induced}_B(b)$, then $b' \notin A\vartheta$.*

Proof. Since an application of a substitution on A preserves its connectedness, there are two possible cases: (i) $A\vartheta \subseteq B \setminus \text{Induced}_B(b)$ or (ii) $A\vartheta \subseteq \text{Induced}_B(b) \setminus \{b\}$. The latter case is not possible because in the definition of H-subsumption, the substitutions were required to respect depth of variables and every variable contained in $\text{Induced}_B(b) \setminus \{b\}$ has depth w.r.t. B greater than 1. Therefore the only possibility is $A\vartheta \subseteq B \setminus \text{Induced}_B(b)$. \square

Lemma 3. *Let A, B be $(\gamma, \mu)_\theta$ -features or pos (γ, μ) -blocks and let ϑ be an H-substitution such that $A\vartheta \subseteq B$. Then if for some $a \in A$, $b \in B$ it holds $a\vartheta = b$ then $\text{Induced}_A(a)\vartheta \subseteq \text{Induced}_B(b)$.*

Proof. Suppose for contradiction that there is some $a' \in \text{Induced}_A(a)$ such that $a'\vartheta \notin \text{Induced}_B(b)$. There is then some longest sequence of atoms a_1, \dots, a, \dots, a' ($a_i \in A$) such that every two consecutive atoms share a variable, which is an output in the first one and an input in the second one. It follows from definition of depth and from definition of H-subsumption that ϑ maps atoms from this sequence to atoms from sequence b_1, \dots, b, \dots, b' , where every two consecutive atoms share a variable, which is an input in the first one and an output in the second one, and b' is an atom in the same depth as a' . One such sequence must obviously exist. If there were more such sequences then B would not be treelike. This is a contradiction with the assumption that $a'\vartheta \notin \text{Induced}_B(b)$ because $b' \in \text{Induced}_B(b)$. \square

Lemma 4. *Let F be a τ_θ -feature or a pos τ -block. If there is an H-substitution ϑ such that $F\vartheta \subseteq F$ and there are atoms $a, b \in F$ and their respective induced pos τ -blocks $A, B \subseteq F$ such that $A\vartheta \subseteq B$ then either $p(A) = p(B)$ or there are pos τ blocks $A', B' \subseteq F$ such that $A \subsetneq A'$ and $B \subsetneq B'$ and $A'\vartheta \subseteq B'$.*

Proof. If $A\vartheta \subseteq B$ then $\mathcal{P}(b) \in F\vartheta$ by application of Lemma 2, from which $\mathcal{P}(a)\vartheta = \mathcal{P}(b)$. If $p(\mathcal{P}(a)) = p(\mathcal{P}(b))$, we are done. Otherwise, since $F\vartheta \subseteq F$ we have $\text{Induced}_F(\mathcal{P}(a))\vartheta \subseteq F$ therefore $\text{Induced}_F(\mathcal{P}(a))\vartheta \subseteq \text{Induced}_F(\mathcal{P}(b))$ by application of Lemma 3 because $\mathcal{P}(a)\vartheta = \mathcal{P}(b)$. \square

Proposition 9 Let B be a pos τ -block and let B^- be a neg τ -block. Then the following holds: (i) B is H-reducible if and only if B contains pos τ -blocks B_1, B_2 such that $B_1 \neq B_2$, $p(B_1) = p(B_2)$ and $B_1 \preceq_H B_2$. (ii) If B is H-reducible, then $B^- \oplus (S \cup \{B\})$ is also H-reducible for any set of pos blocks S .

Proof. **(i) \Rightarrow** Let B_r be H-reduction of B and let θ_1, θ_2 be H-substitutions such that $B_r\theta_1 \subseteq B$ and $B\theta_2 \subseteq B_r$. Substitution $\theta_3 = \theta_2\theta_1$ is a mapping $\theta_3 : \text{vars}(B) \rightarrow \text{vars}(B)$. Since $B\theta_2 \subseteq B_r$, $|B\theta_2| \leq |B_r|$ and consequently $|B\theta_3| \leq |B_r| < |B|$, because applying a substitution to a τ -conjunction cannot increase its size. Therefore there is an atom $a \in B \setminus B\theta_3$ and, by Lemma 2, also a whole pos τ -block $B_1 \subseteq B \setminus B\theta_3$. Thus, there is a pos τ -block B_2 ($B_2 \neq B_1$) such that $B_1\theta_3 \subseteq B_2$. It remains to show that for some such B_1, B_2 , $p(B_1) = p(B_2)$. If $p(B_1) \neq p(B_2)$, then there must be pos τ -blocks G_1, G_2 such that $B_1 \subsetneq G_1$, $B_2 \subsetneq G_2$ and $G_1\theta_3 \subseteq G_2$ (by Lemma 4). For such G_1, G_2 with maximum size, $p(G_1) = p(G_2)$. **(i) \Leftarrow** Let θ be a H-substitution such that $F^+ \setminus B\theta = B_1$, then $B\theta \approx_H B$ and $|B\theta| = |B| - |B_1| < |B|$. **(ii)** This follows directly from (i). \square

Lemma 5. Let F be a (γ, μ) -feature. Then $d_F \leq |\gamma|$.

Proof. Assume $d_F > |\gamma|$. Let θ be a substitution such that $F\theta \subseteq \gamma$ and a_1, \dots, a_{d_F} be a sequence of F 's atoms such that for $1 \leq i < d_F$, a_i contains variable v_i as output and a_{i+1} contains v_i as input. By transitivity of the partial order \prec assumed by Def. 17, it must hold $v_i \prec v_j$ for $1 \leq i < j \leq d_F$. Since $d_F > |\gamma|$, the sequence must contain two atoms a_k and a_l , such that $k < l$, $a_k\theta = a_l\theta$ and thus $v_k\theta = v_l\theta$. Since $k < l$, $v_k\theta \prec v_l\theta$. But given that $v_k\theta = v_l\theta$, this contradicts the assumption of irreflexivity of the order \prec . \square

Proposition 10 Let $\tau = (\gamma, \mu)$ be a template. Then there is only a finite number of τ -features, which are not H-reducible.

Proof. First, we show that for any template, the set P_d of pos blocks with depth at most d is finite. **(i)** For $d = 1$, there is at most $|\gamma|$ pos τ -blocks because any pos τ -block with depth 1 is just a single atom. Therefore P_1 is finite. **(ii)** Let us suppose that we have proved the proposition for maximum depth at most d . We need to show that then it holds also for $d + 1$. We can take all atoms $a \in \gamma$ and for each of them create the set of τ -conjunctions

$$P_{d+1}^a = \{a\theta_a \wedge_{i=1}^{\text{arity}(a)} C_i\theta_i \mid C_i \in 2^{P_d}\} \cup P_d,$$

where θ_a is only meant as *variabilization* of a (i.e. as substitution that replaces constants by suitable variables) and θ_i maps $p(S)$ of every pos τ -block $S \in C_i$ to i -th output argument of $a\theta_a$. Not all elements of sets P_{d+1}^a are correct pos τ -blocks, but it is easy to check that all pos τ -blocks with depth at most $d + 1$ are contained in $\cup_{a \in \gamma} P_{d+1}^a$. To see this, it is important to notice that it suffices to create combinations of pos τ -blocks without repetition since any combination with repetitions would necessarily lead to H-reducible pos τ -blocks (Proposition 9). Now, validity of the proposition follows from the fact that $P_d = \cup_{a \in \gamma} P_d^a$ is finite for any d and that for any template there is a maximum depth of τ -features (Proposition 5). \square

Proposition 11 Algorithm 1 correctly decides whether a τ_θ -feature (pos τ -block) F is H-reducible in time $O(|F|^2)$.

Proof. First, we show, using an induction argument, that (a_i, a_j) gets to the list Open if and only if $\text{Induced}_F(a_i) \preceq_H \text{Induced}_F(a_j)$. **(i)** This is trivial for pos blocks with depth 1. **(ii)** We suppose that the claim holds for depth d . The induction argument implies that $\text{Subsumed}(P_F(a_i^{d+1}), P_F(a_j^{d+1})) = |C_F(l_i^{d+1})|$ if and only if every child c of a_i^{d+1} can be mapped by H-substitution to some child c' of a_j^{d+1} while preserving $I_F(c) = I_F(c')$. Validity of the induction step then follows from this together with $\text{predicate}(a_i^{d+1}) = \text{predicate}(a_j^{d+1})$. Now, from Proposition 9 we know that existence of two pos τ -blocks B_1, B_2 with $p(B_1) = p(B_2)$ contained in a feature F such that $B_1 \preceq_H B_2$ is equivalent to H-reducibility of F , which finishes the proof of correctness of the algorithm.

Clearly, each pair of atoms can be added only once to the list `Open` because it happens only once that $\text{Subsumed}(P_F(a_i), P_F(a_j)) = |C_F(a_i)|$ (because $\text{Subsumed}(P_F(a_i), P_F(a_j))$ is only incremented and never decremented and $|C_F(a_i)|$ is a constant). It follows that the *repeat-until*-loop will run for at most $|F|^2$ steps. Since all the other operations can be performed in $O(1)$ time (in the unit-cost RAM model), we have the bound $O(|F|^2)$. \square

Lemma 6. *Let I be an interpretation, τ a template and let $S_1 = \{B_1, \dots, B_m\}$ and $S_2 = \{C_1, \dots, C_n\}$ be standardized-apart sets of pos τ -blocks such that*

$$\bigcap_{i=1}^m \text{dom}_I(B_i) \subseteq \bigcap_{i=1}^n \text{dom}_I(C_i),$$

then for any pos-neg τ_θ -block or neg τ_θ -block B^-

$$\text{dom}_I(B^- \oplus S_1) \subseteq \text{dom}_I(B^- \oplus S_2).$$

Proof. Let us first consider the case when $d_{B^-}(n(B^-)) = 1$. The only place, where domains of $B_i \in S_1$ ($C_i \in S_2$ respectively) are used, is line 5 in Algorithm 2. Clearly $\text{argDom}_{S_1} = \bigcap_{i=1}^m \text{dom}_I(B_i) \subseteq \text{argDom}_{S_2} = \bigcap_{i=1}^n \text{dom}_I(C_i)$ and consequently also $\text{atomsDom}_{S_1} \subseteq \text{atomsDom}_{S_2}$ and therefore also $\text{dom}_I(B^- \oplus S_1) \subseteq \text{dom}_I(B^- \oplus S_2)$. The general case of lemma may be proved by induction on depth of $n(B^-)$. (i) The case for depth 1 has been already proved. (ii) Let us suppose that the lemma holds for depth d . Now, we may take the pos-neg τ_θ -block $T \subseteq B^-$ which contains $n(B^-)$ such that $(B^- \setminus T) \oplus \{T\} = B^-$ and $d_T(n(B^-)) = 1$, and graft it with S_1 (S_2 , respectively). We have $\text{dom}_I(T \oplus S_1) \subseteq \text{dom}_I(T \oplus S_2)$ and by induction argument finally also $\text{dom}_I(B^- \oplus S_1) = \text{dom}_I((B^- \setminus T) \oplus \{T \oplus S_1\}) \subseteq \text{dom}_I((B^- \setminus T) \oplus \{T \oplus S_2\}) = \text{dom}_I(B^- \oplus S_2)$. \square

Proposition 12 *Let E^+ (E^-) be a set of positive (negative) examples. Let \mathcal{F} be a set of pos τ -blocks with equal types of input arguments and let $\mathcal{B} = \{B_i\}_{i=1}^n \subseteq \mathcal{F}$. Let $\text{dom}_I(B_1) \subseteq \bigcap_{i=2}^n \text{dom}_I(B_i)$ for all $I \in E^+$ ($I \in E^-$) and let $\bigcap_{i=2}^n \text{dom}_I(B_i) \subseteq \text{dom}_I(B_1)$ be true for all $I \in E^-$ ($I \in E^+$). Then for any neg τ -block $B^-: B^- \oplus (\{B_1\} \cup \mathcal{S})$ is E^+ -redundant (E^- -redundant), where $\mathcal{S} \subseteq \mathcal{F}$.*

Proof. We will prove only the case for E^+ -redundancy because the proof for E^- -redundancy is analogous. Let us first ignore the set \mathcal{S} . Let B^- be a neg τ -block. By application of Lemma 6, if $\text{dom}_I(B_1) \subseteq \bigcap_{i=2}^n \text{dom}_I(B_i)$ for all $I \in E^+$, then $\text{dom}_I(F^- \oplus \{B_1\}) \subseteq \text{dom}_I(F^- \oplus \{B_2, \dots, B_n\})$ for all $I \in E^+$. Similarly, if $\bigcap_{i=2}^n \text{dom}_I(B_i) \subseteq \text{dom}_I(B_1)$ for all $I \in E^-$, then $\text{dom}_I(B^- \oplus \{B_2, \dots, B_n\}) \subseteq \text{dom}_I(B^- \oplus \{B_1\})$ for all $I \in E^-$. Therefore if $I \models F^- \oplus \{B_1\}$, then $I \models F^- \oplus \{B_2, \dots, B_n\}$ for all $I \in E^+$ and similarly if $I \models B^- \oplus \{B_2, \dots, B_n\}$, then $I \models B^- \oplus \{B_1\}$ for all $I \in E^-$. This means that $B^- \oplus \{B_1\}$ must be E^+ -redundant. Now, we consider also the set $\mathcal{S} = \{H_1, \dots, H_o\}$. Notice that $\text{dom}_I(B_1) \cap_{i=1}^o \text{dom}_I(H_i) \subseteq \bigcap_{i=2}^n \text{dom}_I(B_i) \cap_{i=1}^o \text{dom}_I(H_i)$ must hold for all positive examples and $\bigcap_{i=2}^n \text{dom}_I(B_i) \cap_{i=1}^o \text{dom}_I(H_i) \subseteq \text{dom}_I(B_1) \cap_{i=1}^o \text{dom}_I(H_i)$ must hold for all negative examples. The rest of the proof is obvious; it is an analogy of the argument used to prove the first part of the proposition. \square

Proposition 13 *Let $\mathcal{F} = \{B_i\}_{i=1}^n$ be a set of pos τ -blocks with equal types of input arguments such that no two pos τ -blocks in the set \mathcal{F} have equal domains for all examples. Let \mathcal{F}' be a set of pos τ -blocks obtained from \mathcal{F} by repeatedly removing redundant pos τ -blocks. Set \mathcal{F}' is unique.*

Proof. Let \prec_R be a relation defined as follows: $B \prec_R C$ if and only if $\text{dom}_I(B) \subseteq \bigcap_{k \in A} \text{dom}_I(B_k) \cap \text{dom}_I(C)$ for all $I \in E^+$ and $\bigcap_{k \in A} \text{dom}_I(B) \cap \text{dom}_I(C) \subseteq \text{dom}_I(B)$ for all $I \in E^-$ and $B \neq C$ and $\forall i: B \neq B_i$. It is not hard to check that \prec_R is a partial order (also due to the fact that no two elements of \mathcal{F} have identical domains). Set \mathcal{F}' contains all maximum elements w.r.t. \prec_R and is therefore unique. \square

Proposition 14 *Let $\tau = (\gamma, \mu)$ be a template and let $E = E^+ \cup E^-$ be the set of examples. Further, let \mathcal{F}_τ be the set of all non- H -reducible τ -features and let $\mathcal{F}_{\text{RelF}}$ be the set of conjunction of atoms constructed by RELF. Then $\mathcal{F}_{\text{RelF}} \subseteq \mathcal{F}_\tau$ and for every τ -feature $F_\tau \in \mathcal{F}_\tau$, which is not strictly redundant, there is a feature F_{RelF} such that $\text{ext}_E(F_{\text{RelF}}) = \text{ext}_E(F_\tau)$.*

Proof. (Sketch) It is not hard to see that $\mathcal{F}_{\text{RELf}} \subseteq \mathcal{F}_\tau$. For each conjunction of atoms $F \in \mathcal{F}_{\text{RELf}}$, we just construct a substitution θ such that θ maps each atom $a \in F$ to an atom from γ that was used when a was created in line 2 in Algorithm 4. Then it is not hard to show that such a substitution is consistent (i.e. there are no two conflicting substitutions of one variable) and that $F\theta \subseteq \gamma$ and that also the conditions on input and output arguments stated in Definition 19 are satisfied.

In order to justify the proposition, we need to show that if RELF did not remove redundant pos τ -blocks or pos τ -blocks that do not cover any positive example, it would construct all τ -features contained in \mathcal{F}_τ . The case with filtering of redundant pos τ -blocks will follow from Propositions 9 and 12. First, recall that from earlier propositions we know that all features are treelike and $|\mathcal{F}_\tau|$ is finite. For contradiction, let us assume that there is a substitution θ and a non-H-reducible τ_θ -feature $F \in \mathcal{F}_\tau$ such that $F \notin \mathcal{F}_{\text{RELf}}$. Let B be a shortest pos τ_θ -block such that $F = B^- \oplus \{B\}$ and such that B was not constructed by RELF (i.e. the shortest block which caused that F could not be constructed). If no such block exists, let us set $B = F$. Further, let $a \in \gamma$ be an atom that corresponds to B 's root. By a brief analysis of the algorithm RELF, we may conclude that if all pos τ_θ -blocks, which can be composed with a using the graft operator, had been generated, then procedure $\text{Combine}(\text{Atom}, \text{PosBlocks}, E)$ would have also generated B on line 5 of Algorithm 4. Since the necessary blocks must be generated at some point in the feature construction process (because otherwise we would have a contradiction with the minimality of B), the only possibility is that they were generated only after the atom a had been processed. This possibility is, however, ruled out by the requirement on the partial irreflexive order on constants in γ given in Definition 17 (recall that $F\theta \subseteq \gamma$) and by the fact that RELF sorts the atoms from γ according to this order before it starts the feature construction process. Thus we see that B and consequently also F had to be constructed by RELF, which finishes the proof. \square

5.14 NOTES ON COMPLEXITY OF TEMPLATE-BASED FEATURE CONSTRUCTION

In this section, we elaborate the complexity of constructing all syntactically correct features. We will be interested in the complexity of construction of treelike and non-treelike features constrained using the template-bias mechanism presented in this Chapter. We will ignore extension computations and the fact that it is possible to prune some features, which do not cover any example, i.e. we will focus solely on the generation of features constrained by templates. We believe it is useful to elaborate complexity of this problem because the setting with syntactical constraints given by templates seems to be very useful for setting a language bias for real-life problems.

We will use unary representation of numbers used to bound sizes of features. This is because a number n can be represented by $O(\log n)$ bits, which could allow existence of features with size exponential in the input size (in the combined size of the template and of the binary representation of n). By choosing to use the unary representation of numerical parameters, we will be dealing with the so called strong NP-completeness [110].

5.14.1 Negative Results

We start with a negative result which indicates that constructing non-treelike features constrained by the template-based language bias and with size bounded by a given number is a computationally hard problem.

Lemma 7. *Let $\tau = (\gamma, \mu)$ be a template and $n \in \mathbb{N}$ be a number represented in unary notation (i.e. number n is represented by a string $111 \dots 1$ with n ones). The problem of deciding, whether there is at least one feature correct w.r.t. τ and n , is in NP.*

Proof. (Sketch) It suffices to show how to construct a polynomial-sized certificate and a polynomial-time verifier. The certificate will be a pair (F, θ) , where F is a conjunction of literals such that

$|F| \leq n$ and θ is a substitution. Clearly, the certificate has size polynomial in $|\tau|$ and in the unary representation of n .

Now, it remains to show how we can verify correctness of the solution in polynomial time, but this is easy. First, we check that $|F| \leq n$ and that $\text{lits}(F\theta) \subseteq \gamma$. Then we check whether every variable appears both as an input and as an output and whether every variable appears only once as an output, which is easy because θ uniquely determines, which variable appearance is an input and which is an output. If none of these checks fails, we may output *yes* and accept the feature. \square

Lemma 8. *Let $\tau = (\gamma, \mu)$ be a template and $n \in \mathbb{N}$ be a number represented in unary notation (i.e. number n is represented by a string $111 \dots 1$ with n ones). The problem of deciding, whether there is at least one feature correct w.r.t. τ and n , is NP-hard.*

Proof. (Sketch) We will prove this proposition by reduction from the graph coloring problem. Let $G = (V, E)$ be the graph to be colored and let the set of *colors* be $\{\text{red, green, blue}\}$. First, we make the edges oriented such that an edge between two vertices v_i, v_j will be pointing from v_i to v_j if $i < j$. The template τ will be constructed as follows. The first declared predicate will be

$$\text{graph}(-e_1, -e_2, \dots, -e_{|E|}, -v_1, \dots, -v_{|V|})$$

where each e_i will correspond to one edge $e \in E$ ($n = |E|$). We add the six following declared predicates (called *edge predicates*)

$$\begin{aligned} &\text{edge}_{rb}(+\text{rout}_{e_k}, +e_k, -\text{bin}_{e_k}), \text{edge}_{br}(+\text{bout}_{e_k}, +e_k, -\text{rin}_{e_k}), \\ &\text{edge}_{rg}(+\text{rout}_{e_k}, +e_k, -\text{gin}_{e_k}), \text{edge}_{gr}(+\text{gout}_{e_k}, +e_k, -\text{rin}_{e_k}), \\ &\text{edge}_{bg}(+\text{bout}_{e_k}, +e_k, -\text{gin}_{e_k}), \text{edge}_{gb}(+\text{gout}_{e_k}, +e_k, -\text{bin}_{e_k}) \end{aligned}$$

for each edge $e_k \in E$ pointing from v_i to v_j . Next, for each vertex v_i we add three declared predicates (called *vertex predicates*)

$$\begin{aligned} &\text{red}_i(+v_i, +\text{rin}_{e_{i_1}}, \dots, +\text{rin}_{e_{i_n}}, -\text{rout}_{e_{j_1}}, \dots, -\text{rout}_{e_{j_m}}) \\ &\text{green}_i(+v_i, +\text{gin}_{e_{i_1}}, \dots, +\text{gin}_{e_{i_n}}, -\text{gout}_{e_{j_1}}, \dots, -\text{gout}_{e_{j_m}}) \\ &\text{blue}_i(+v_i, +\text{bin}_{e_{i_1}}, \dots, +\text{bin}_{e_{i_n}}, -\text{bout}_{e_{j_1}}, \dots, -\text{bout}_{e_{j_m}}) \end{aligned}$$

where each of the above declared predicates corresponds to a vertex v_i and v_{i_1}, \dots, v_{i_n} correspond to vertices from which an edge points to v_i and vertices v_{j_1}, \dots, v_{j_m} correspond to vertices to which an edge points from v_i . Finally, we set the maximum feature size $n = 1 + |V| + |E|$.

It is trivial to check that the above construction is polynomial-time in the size of G . Now, it remains to show that a feature $F, |F| \leq 1 + |V| + |E|$ exists, which complies to the above constructed template, if and only if a 3-coloring of G exists.

(\Rightarrow) First, notice that there must be exactly one edge predicate for each $e \in E$ and exactly one vertex predicate for each $v \in V$ in a correct feature F . Otherwise, there would be an unsatisfied output v_i or e_i . Therefore each vertex has exactly one color represented by a vertex predicate. Furthermore, colors of two adjacent vertices must be different due to types of input and output arguments of the respective vertex predicates. So, the coloring given by the vertex predicates represents a correct coloring of G .

(\Leftarrow) If we have a coloring of $G = (V, E)$, we may construct a correct feature with size bounded $|F| = 1 + |V| + |E|$. We add one literal $\text{graph}(-e_1, -e_2, \dots, -e_{|E|}, -v_1, \dots, -v_{|V|})$. Next, we add one vertex literal for each $v \in V$ (choosing the particular predicate according to the coloring of the given vertex). Finally, we add one edge literal for each $e \in G$ (again, we choose the particular predicate according to the coloring of the vertices connected by this edge). It is easy to check that this corresponds to a correct feature. \square

Proposition 15. *Let $\tau = (\gamma, \mu)$ be a template and $n \in \mathbb{N}$ be a number represented in unary notation (i.e. number n is represented by a string $111 \dots 1$ with n ones). The problem of deciding, whether there is at least one feature correct w.r.t. τ and n (called feature existence problem), is NP-complete.*

Proof. Follows directly from Lemmas 7 and 8. \square

5.14.2 Positive Results

The negative results presented in the previous section show that generating non-treelike features smaller than a given size limit and constrained by the template-based language bias mechanism is a computationally hard problem. On the other hand, we have seen in Section 5.9 where the algorithm HiFi was presented that there is no such problem when one works only with treelike features. However, nothing was said about the runtime complexity of HiFi or RelF. Here we show that when one considers only the task of generating all syntactically correct features then HiFi works in output-polynomial time.

Proposition 16. *Let $\tau(n) = (\gamma(n), \mu(n))$ be a template such that $|\gamma(n)| = O(n^c)$ for some $c \geq 1$, then the feature construction part of HiFi without extension computation and redundancy filtering can construct all correct treelike features w.r.t. τ and n in output-polynomial time.*

Proof. Any pos block, which is stored in the set PosBlocks, and any combination of pos blocks, which is generated by procedure BuildCombinations(PosBlocks, Type, Examples), is used at least once in the resulting set of generated features, which is guaranteed by the fact that we can check whether there is at least one correct sufficiently small feature containing a given pos block (see discussion in Section 5.9). Thus at any time, we can bound the number of pos blocks in both of these sets by $n \cdot P(n)$, where $P(n)$ is number of correct features, because no feature can contain more than n pos blocks (recall that we say that a pos block B is contained in a feature if and only if $B \subset F$ and $F \setminus B$ is a neg block).

Brief combinatorial reasoning implies that generating pos blocks with root equal to a given predicate p using already generated pos blocks takes time polynomial in the number of pos blocks just being generated. In more detail: combinations of already generated pos blocks are created iteratively. At each step the new combinations of pos blocks are combined with single pos blocks from the set PosBlocks; each such step takes time at most quadratic in the size of the already generated combinations and, always, every generated combination of pos blocks appears at least in one correct feature. Thus generation of combinations of pos blocks is polynomial in the total number of features correct w.r.t. τ and n . An analogical reasoning can be applied also to generation of pos blocks with a given predicate of their root from these combinations.

As there are only $O(n^c)$ declared predicates, time complexity of HiFi is polynomial in $P(n)$. \square

It is also important to understand what Proposition 16 does not say. It does not say that HiFi runs in output-polynomial time when given a set of learning examples and a template τ . It only says that it runs in output-polynomial time for the task of generating the set of all syntactically correct non-H-reducible features. Nevertheless, to our best knowledge, there have been only results about output-polynomiality of algorithms which generate all frequent patterns (of various sorts, e.g. graph patterns) in the literature but no results about output-polynomiality for the problem of finding complete sets of non-redundant features because the latter is actually a much more difficult problem.

The algorithms RelF and HiFi, presented in the previous chapter, are not able to work efficiently when the input data contains a large amount of information in numerical form. Here, we describe another new algorithm for construction of treelike polynomial features called Poly which is suitable for such domains. The algorithm Poly is based on the feature-construction algorithm HiFi from which it differs mainly in the way it filters redundant building blocks of features and in the way features are evaluated.

This chapter is organized as follows. We start by presenting a new fast algorithm for evaluation of treelike polynomial features in Section 6.1. Then, in Section 6.2, we describe a similar fast algorithm for bounded-treewidth generalized multivariate aggregation features with a fixed number of distinguished constants. Since the problem of evaluating bounded-treewidth generalized multivariate aggregation features is NP-hard as we showed in Chapter 4, the restriction on the fixed number of distinguished variables is unavoidable in this algorithm as long as it should be polynomial-time (and as long as $P \neq NP$). In Section 6.3, we explain how redundant $\text{pos } \tau_\theta$ -blocks of polynomial features can be detected and, in Section 6.4, we describe a method for the same problem for generalized multivariate aggregation features. In Section 6.5 we briefly outline how the pieces of the *puzzle* presented in Sections 6.1 and 6.3 can be assembled and combined with the algorithm HiFi, which gives us the algorithm Poly as a result. We evaluate the algorithm Poly experimentally in Section 6.6. Finally, Section 6.7 concludes the chapter. Proofs of propositions are located in Section 6.8.

Note that in this chapter we continue using the formalism of τ -features introduced in the previous chapter. Therefore we assume in this chapter that the reader is familiar with the notions like *template*, *pos block*, *neg block* and *domain*.

6.1 EVALUATION OF POLYNOMIAL FEATURES

An important property of polynomial features is that their value can be computed *quite* efficiently. The problem is NP-hard for polynomial features in general, which can be shown by reduction from θ -subsumption hardness, but there are tractable subclasses which may be solved efficiently. Here, we present an algorithm for evaluation of bounded-treewidth polynomial features which runs in polynomial time (first, we describe the algorithm only for treelike features and then outline how it can be extended for bounded-treewidth features). The basic ideas of the algorithm can be summarized as follows. Let us have a treelike feature F with k distinguished variables, a monomial feature

$$M = (F, (d_1, \dots, d_k))$$

and an example

$$e = (H, \vec{\theta}).$$

For simplicity, we assume that any literal $l \in F$ may contain at most one distinguished variable¹. We want to compute the value $M(e)$. We start by picking a literal $l \in F$ containing a distinguished variable R_1 and ground all its variables using a substitution $\vartheta : \text{vars}(l) \rightarrow \text{constants}(c)$ so that $e \models F\vartheta$. We then create a new auxiliary monomial

$$M_\vartheta = (F\vartheta, (d_1, \dots, d_k)).$$

¹ This is without loss of generality because any representation with demanding more than one distinguished variable per literal can be rewritten into a representation where there is always only one distinguished variable per literal.

Algorithm 5 An algorithm for computing value $M(e)$ of a monomial feature $M = (F, (d_1, \dots, d_k))$ for a treelike τ_θ -feature (or a treelike pos τ_θ -block) F and an example $e = (H, \vec{\theta})$

Procedure: Eval(M, e)

- 1: $SP \leftarrow \square$ /* An associative array of sample parameters */
- 2: $\text{atomsDom} \leftarrow \{a \in H \mid \text{pred}(a) = \text{pred}(\text{root}(F)) \wedge (H \models a)\}$
- 3: /* First, we fill in the array SP with sample parameters of individual atoms that could be substituted for the root. */
- 4: **for** $\forall a \in \text{atomsDom}$ **do**
- 5: $r_{i_1}, r_{i_2}, \dots, r_{i_{d_i}} \leftarrow$ distinguished constants contained in $\text{root}(F)$ (indexed by indexes of the corresponding distinguished variables from $\text{root}(F)$)
- 6: $SP[a] \leftarrow (a, r_{i_1}^{d_{i_1}} \cdot r_{i_2}^{d_{i_2}} \dots r_{i_{d_i}}^{d_{i_{d_i}}}, 1)$
- 7: **end for**
- 8: **for** \forall output variables out_i in $\text{root}(F)$ **do**
- 9: $\text{Children} \leftarrow$ all pos τ_θ -blocks Child such that $\text{out}_i = p(\text{Child})$
- 10: **for** $\forall \text{Child} \in \text{Children}$ **do**
- 11: $SP_{\text{Child}} \leftarrow \square$ /* An associative array of sample parameters */
- 12: /* We just store the sample parameters of Child into the associative array SP_{Child} . */
- 13: **for** $(x, v, n) \in \text{Eval}((\text{Child}, (d_1, \dots, d_k)), e)$ **do**
- 14: $SP_{\text{Child}}[x] \leftarrow (x, v, n)$
- 15: **end for**
- 16: **for** $\forall a \in \text{atomsDom}$ **do**
- 17: **if** SP contains key a **then**
- 18: **if** SP_{Child} contains key $\text{arg}_i(a)$ **then**
- 19: $SP[a] \leftarrow SP[a] \otimes SP_{\text{Child}}[\text{arg}_i(a)]$
- 20: **else**
- 21: Remove the key-value pair with key a from SP
- 22: **end if**
- 23: **end if**
- 24: **end for**
- 25: **end for**
- 26: **end for**
- 27: **return** $\bigoplus_F \{sp \mid sp \text{ is contained in the array } SP\}$

The problem of computing $M_\vartheta(e)$ can be decomposed as

$$M_\vartheta(e) = (\vec{\theta}_{I(\vartheta)})^{d_1} \cdot \prod_i M_i(e) \quad (7)$$

where M_1, \dots, M_m are connected sub-features of $F\vartheta$ which arise when we remove literal $l\vartheta$ from $F\vartheta$ and $\vec{\theta}_{I(\vartheta)}$ is the *value* of the distinguished variable R_1 contained in l corresponding to the substitution ϑ . This follows from Proposition 5.

The value $M(e)$ can be then computed as

$$M(e) = \frac{1}{\sum_{\vartheta \in \Theta} \alpha_\vartheta} \sum_{\vartheta \in \Theta} \alpha_\vartheta M_\vartheta(e) \quad (8)$$

where $\Theta = \{\vartheta : \text{vars}(l) \rightarrow \text{constants}(c) \mid e \models F\vartheta\}$ is the set of all true groundings of literal l and α_ϑ are the numbers of true groundings of $F\vartheta$.

Now, we describe the algorithm in more detail. The basic elements of the algorithm are operations on so-called *sample-parameters*. By *sample parameters* we mean a 3-tuple (x, v, n) . Here x can be either an empty set, a literal or a term, v is a real number and n is a natural number. Next,

we define a multiplication operation for combining sample parameters. Let $A = (x, v, n_A)$ and $B = (y, w, n_B)$ be sample parameters. Then we define $A \otimes B$ as

$$A \otimes B = (x, v \cdot w, n_A \cdot n_B).$$

Let F be a pos τ_θ -block. Let $A = (x, v, n_A)$ and $B = (y, w, n_B)$ be sample parameters where $x \in F$ and $y \in F$ are first-order-logic literals such that $p(x) = p(y)$ (here, $p(x)$ and $p(y)$ are input variables of the literals x and y in the pos τ_θ -block F). Then we define $A \oplus_F B$ as

$$A \oplus_F B = \left(p(x), \frac{1}{n_A + n_B} (n_A \cdot v + n_B \cdot w), n_A + n_B \right).$$

If F is a τ_θ -feature then we define

$$A \oplus_F B = \left(\emptyset, \frac{1}{n_A + n_B} (n_A \cdot v + n_B \cdot w), n_A + n_B \right).$$

Let F be a pos τ_θ -block. Let

$$\mathcal{X} = \{(x_1, v_1, n_1), \dots, (x_k, v_k, n_k)\}.$$

Next, let $\mathcal{X}[t]$ denote the set of all sample parameters $(x, \dots) \in \mathcal{X}$ for which $p(x) = t$ (where $p(x)$ is the input variable of the literal x in the pos τ_θ -block F). Then $\bigoplus_F \mathcal{X}$ is defined as follows.

$$\bigoplus_F \mathcal{X} = \{(y_1, w_1, n_{w_1}), \dots, (y_m, w_m, n_{y_m})\}$$

where $(y_i, w_i, n_{y_i}) = x_1 \oplus_F x_2 \oplus_F \dots \oplus_F x_p \oplus_F (\emptyset, 0, 0)$ for $\{x_1, \dots, x_p\} = \mathcal{X}[y_i]$.

The pseudocode of the algorithm is shown in Algorithm 5. Its correctness follows from the discussion in the beginning of this section. The procedure `Eval` computes the resulting value of the polynomial feature M using Eq. 8 from the values computed and saved in the associative array SP . This associative array SP contains values of monomials $M_\vartheta = (F\vartheta, (d_1, \dots, d_k))$ indexed by $\text{root}(F)\vartheta$ (where ϑ is a substitution affecting only the variables in $\text{root}(F)$). It is not too difficult to see that the procedure `Eval` computes these values exactly according to Eq. 7.

The time-complexity of the algorithm is $\mathcal{O}(|F| \cdot |e| \cdot k)$ where k is the number of distinguished variables. The procedure `Eval` can be imagined as proceeding from leaves to root of the feature. At each step, at most $|e|$ literals must be processed in the loop on line 4 and the complexity of each iteration of this loop is $\mathcal{O}(k)$ where k is the number of distinguished variables. The \bigoplus_F operation can be performed in time $\mathcal{O}(|e| \cdot k)$. The for-loop on line 8 contains nested loops. However, the nested loop on line 10 is executed at most once for each literal $l \in F$ (because any pos block has at most one parent) and the inner-most loops are always executed $\mathcal{O}(|e|)$ -times for every $l \in F$. So the overall complexity of the algorithm is indeed $\mathcal{O}(|F| \cdot |e| \cdot k)$. Note that, here, we assume that we have associative arrays (hash-tables) with $\mathcal{O}(1)$ -time operations for *insert*, *delete* and *get*.

The outlined algorithm for computing values $M(e)$ of treelike features can be used to efficiently compute values of monomial features which are not treelike but have bounded tree-width. This can be done by computing a tree decomposition [101] of width k of the feature and then by applying the algorithm for treelike features on the tree decomposition where only one occurrence of each distinguished variable is retained.

6.2 EVALUATION OF GENERALIZED MULTIVARIATE AGGREGATION FEATURES

For completeness, in this section we also describe an efficient algorithm for a constrained class of generalized multivariate aggregation features. The rather well-known fact that the number of true groundings of a bounded-treewidth conjunctive-query can be computed in polynomial

time [101] may be used to construct an efficient algorithm for computing values of generalized multivariate aggregation features under the assumption that the number of distinguished variables is fixed (and small). The algorithm works as follows.

Given a multivariate aggregation feature

$$M = (F, f(R_1, \dots, R_k))$$

(where $f(R_1, \dots, R_k)$ is an aggregation function) and an example

$$e = (H, \vec{\theta}),$$

it creates a set of all k -tuples of distinguished constants occurring in H . The number of these tuples grows polynomially with the size of e (exponentially with k , but k is fixed). Then it substitutes the values of every tuple to the distinguished variables in M and counts (in polynomial time) the number of true groundings of the new *restricted* F w.r.t. e . Each tuple $\theta = (r_1, \dots, r_k)$ gives us a vector of real numbers $v_\theta = (s_1, \dots, s_k)$. Then the algorithm can compute the value $f(s_1, \dots, s_k)$ for each of these tuples. Finally, $M(e)$ can be computed as the weighted average of these values where the weights are the numbers of true groundings of $F\theta$ obtained for the respective tuples θ .

Example 24. We illustrate evaluation of multivariate aggregation features on the example of a monomial feature (for which we could also use the more efficient algorithm presented in Section 6.1). Let

$$F = a(X, R_1) \wedge e(X, Y) \wedge a(Y, R_2)$$

be a feature and

$$e = (\{a(a, r_1), e(a, b), e(b, a), a(b, r_2), e(b, c), a(c, r_3)\}, (2, 2, 3))$$

be an example. Next, let

$$M = R_1 \cdot R_2^2$$

be a monomial feature. There are the following 9 2-tuples of distinguished constants in e :

$$\{(r_1, r_1), (r_1, r_2), (r_1, r_3), (r_2, r_1), (r_2, r_2), (r_2, r_3), (r_3, r_1), (r_3, r_2), (r_3, r_3)\}.$$

Only the tuples (r_1, r_2) , (r_2, r_1) and (r_2, r_3) correspond to non-empty sets of groundings. The value of $M(e)$ can then be computed as

$$M(e) = \frac{1}{3} (1 \cdot 2 \cdot 2^2 + 1 \cdot 2 \cdot 2^2 + 1 \cdot 2 \cdot 3^2) = \frac{34}{3}.$$

As we have explained in Section 4.6, the problem of evaluation of multivariate aggregation features is NP-hard even for features with treewidth 1. The algorithm presented here shows that a polynomial time algorithm exists for bounded-treewidth features which have a *fixed number of distinguished variables*.

6.3 REDUNDANCY OF POLYNOMIAL FEATURES

In this section, we extend the methods for efficient filtering of *redundant* features during feature construction for the case of polynomial features.

Definition 31 (Redundancy of polynomial features). *Let \mathcal{E} be a set of examples and \mathcal{F} be a set of polynomial features. We say that $M_F = (F, (d_1, \dots, d_k)) \in \mathcal{F}$ is redundant w.r.t. \mathcal{E} if there is another feature $M_G = (G, (e_1, \dots, e_l)) \in \mathcal{F}$ such that for all examples $e \in \mathcal{E}$, it holds $M_F(e) = c \cdot M_G(e)$.*

This is a reasonable definition of redundancy. It is quite natural that features which give rise to linearly dependent attributes are considered redundant.

Definition 31 tells us when a feature is redundant but it gives us no hints how to recognize that a pos block will remain redundant no matter with what the feature construction algorithm will combine it. The next proposition provides means for detecting redundant pos blocks. It is analogical to Proposition 12.

Proposition 17. *Let B_1 and B_2 be pos τ_θ -blocks such that $\text{vars}(B_1) \cap \text{vars}(B_2) = \emptyset$ and let $M_1 = (B_1, (c_1, \dots, c_k))$ and $M_2 = (B_1, (d_1, \dots, d_l))$. If, for all examples $e = (H, \vec{\theta}) \in \mathcal{E}$, it holds $\text{Eval}(M_1, e) = \text{Eval}(M_2, e)$ then $N_1 = (B^- \oplus B_1, (c_1, \dots, c_k))$ and $N_2 = (B^- \oplus B_2, (d_1, \dots, d_l))$ are redundant (relatively to each other) for any neg τ_θ -block B^- .*

The above proposition can be used in the feature-construction algorithm to detect which pos blocks will always give rise to redundant polynomial features no matter with what the feature-construction algorithm combines them. This is a generalization of results on monotonicity of redundancy from Chapter 5 to settings with polynomial relational aggregation. As a consequence of this, the search space of features can be drastically reduced.

6.4 REDUNDANCY OF GENERALIZED MULTIVARIATE AGGREGATION FEATURES

Redundant blocks can be detected also for generalized multivariate aggregation features. The only problem is that this seems to be computationally harder than for polynomial features.

Definition 32 (Redundancy of multivariate aggregation features). *Let \mathcal{E} be a set of examples and \mathcal{F} be a set of multivariate aggregation features. We say that $M_F = (F, f) \in \mathcal{F}$ is redundant w.r.t. \mathcal{E} if there is another feature $M_G = (G, g) \in \mathcal{F}$ such that for all examples $e \in \mathcal{E}$, it holds $M_F(e) = c \cdot M_G(e)$.*

It is not difficult to see that if $M_F = (F, f) \in \mathcal{F}$ is a feature and there is another feature $M_G = (G, g) \in \mathcal{F}$ such that $\mathcal{S}(F, e) = \mathcal{S}(G, e)$ for all $e \in \mathcal{E}$ then M_F is redundant.

Now we show how we can detect that a pos τ_θ -block will remain redundant no matter with what the feature construction algorithm would combine it. The next proposition provides means for detecting such redundant pos blocks.

Proposition 18. *Let B_1 and B_2 be pos τ_θ -blocks such that $\text{vars}(B_1) \cap \text{vars}(B_2) = \emptyset$. Let $F_1 = B^- \oplus B_1$ and $F_2 = B^- \oplus B_2$ be features. If, for all examples $e = (H, \vec{\theta})$ in a given dataset \mathcal{E} , it holds $\text{dom}_H(B_1) = \text{dom}_H(B_2)$ and if there is a real number c such that $\mathcal{S}(B_1\theta_{1,t}, e) = \mathcal{S}(B_2\theta_{2,t}, e)$ for all examples $e \in \mathcal{E}$ and all substitutions $\theta_{1,t} = \{p(B_1)/t\}$ and $\theta_{2,t} = \{p(B_2)/t\}$ where $t \in \text{dom}_H(B_1) = \text{dom}_H(B_2)$ then $M_{F_1} = (F_1, f)$ and $M_{F_2} = (F_2, f)$ are redundant for any function f .*

The main problem of the filtering of redundant pos blocks based on Proposition 18 is that it requires comparing sample sets whose size scales as $O(|e|^k)$ where k is the number of distinguished variables. In the case of polynomial features, it is possible to filter individual monomial features $(B, (d_1, \dots, d_k))$ where B is a pos τ_θ -block efficiently. Naturally, a problem is that if we want to construct monomial features exhaustively, there might be up-to $O((k+d)^{k-1})$ non-redundant monomial features. Although the term $(k+d)^{k-1}$ may seem scary, k and d are typically very small which makes the filtering approach for polynomial features based on Proposition 17 better in practice.

6.5 CONSTRUCTION OF FEATURES

Since the previous sections introduced a way redundancy can be used to prune polynomial features, it is now quite easy to extend the algorithm HiFi described in Section 5.9 to be able to construct polynomial features. The new algorithm is called Poly. The main difference of Poly and HiFi is that where HiFi uses *domains*, Poly uses sets of sample parameters computed by

Algorithm 5. Otherwise the two algorithms are almost identical. The only other difference is filtering of reducible features.

The algorithm HiFi uses filtering of reducible features based on H-reducibility (Section 5.4) but this is not suitable for construction of polynomial (or multivariate aggregation features in general) as the next example indicates.

Example 25. Let us have a feature

$$F = \text{regulates}(A, B) \wedge \text{regulates}(A, C) \wedge \text{gene}(B, R_1) \wedge \text{gene}(C, R_2).$$

This feature could e.g. be used to estimate moments of distributions of *expression-levels* of pairs of genes regulated by one common transcription factor. It is true that F is not perfect for this task as it also assumes pairs containing one gene twice but, still, the feature F can provide us valuable information and it can also serve well in a classification setting. Despite that, systems which prune θ -reducible features would throw this feature away.

It is possible to remedy the problems caused by θ -reduction while keeping the benefits of it by using a refined definition which we call r -reduction. We say that a feature F is r -reducible if there exists a feature $G \subseteq F$ and a substitution ϑ such that $F\vartheta \subseteq G$, $|F| < |G|$ and $R_i\vartheta \neq R_j\vartheta$ for any $i \neq j$ where R_i and R_j are distinguished variables. Poly uses a modification of H-reduction compatible with the definition of r -reduction. R -reducible polynomial features do not have to be redundant but it still seems better to discard them.

6.6 EXPERIMENTS

We assessed performance of the algorithm Poly and indirectly also the usefulness of polynomial features in general in two sets of experiments. In the first set of experiments described in Section 6.6.1, we employed polynomial features in the framework of propositionalization. In the second set of experiments described in Section 6.6.2, we used Poly for construction of novel definitions of gene sets for set-level classification of microarray data. In both sets of experiments, Poly was able to obtain good results using only a limited amount of information.

6.6.1 Polynomial Features for Propositionalization

We evaluated performance of the polynomial-feature-based method in three relational learning domains. We compared it with Tilde [8] and with results from literature. We performed experiments with treelike polynomial relational features with degree one, two and three in order to evaluate impact of degree of monomials on predictive accuracy. For each dataset, we used two types of relational descriptions with different complexity. We used random forest classifiers with 100, 500 and 1000 trees (see Table 20).

Our first set of experiments was done on the well-known Mutagenesis dataset [112], which consists of 188 organic molecules marked according to their mutagenicity. We performed two experiments in this domain. In the first experiment, we used only information about bonds and their types (*single, double, triple, resonant*) and information about charge of atoms, but not about their types. In the second experiment, we also added information about atom types. The accuracies obtained by our method (Table 20) are consistently higher than the best accuracy 86% achieved by Tilde in [8]. The best results are obtained for monomial features of degree 3.

Our second set of experiments was performed on the NCI 786 dataset which contains 3506 molecules labelled according to their ability to inhibit growth of renal tumors. Again we performed two experiments in this domain. In the first experiment, we used only information about bonds and their types and information about charge of atoms and in the second experiment we also added information about atom types. Monomials of degree 3 turned out to be best for the first representation whereas monomials of degree 2 performed best for the second representation. Tilde did not perform well on this dataset, so at least, we compared our results with results

	Degree 1	Degree 2	Degree 3
	100/500/1000	100/500/1000	100/500/1000
Muta - Charge	88.8/88.3/88.3	88.8/88.8/88.3	89.9/89.9/89.4
Muta - Atoms + Charge	87.8/88.3/88.3	88.3/88.8/88.8	89.9/89.9/89.9
NCI 786 - Charge	61.0/61.0/61.0	66.5/66.5/66.8	67.2/68.0/68.0
NCI 786 - Atoms + Charge	70.3/70.1/69.9	70.5 / 70.8/70.8	70.6/70.2/70.4
PD138/NB110 - Charge	84.7/82.3/82.3	81.0/79.5/81.0	81.0/79.8/79.8
PD138/NB110 - Propensities	82.7/84.7/84.3	83.9/84.7/84.3	85.1/85.5/85.5

Table 7: Accuracies estimated by 10-fold cross-validation using transformation-based learning with monomial features and random forests for degrees of monomial features: 1, 2 and 3.

reported in Chapter 5 for kFOIL (63.1), nFOIL (63.7) and RelF (69.6). The accuracies obtained with monomial features for the *atoms + charge* representation were consistently higher than these results.

Our third set of experiments dealt with prediction of DNA-binding propensity of proteins. Several computational approaches have been proposed for the prediction of DNA-binding function from protein structure. It has been shown that electrostatic properties of proteins are good features for predictive classification. A recent approach in this direction is the method of Szilágyi and Skolnick [120] who created a logistic regression classifier based on 10 features also including electrostatic properties. Our first model for predicting whether a protein binds to DNA used only distributions of charged amino acids in fixed-size windows and the secondary structure of the proteins. In our second model, we added also information about average *propensity* of amino acids in the fixed-size windows to bind to DNA which had been measured by Sathyapriya et al. [106]. The accuracies obtained by our method on the second model were consistently higher than 81.4% accuracy obtained by Szilágyi and Skolnick with logistic regression or 82.2% that we obtained using random forest on Szilágyi’s and Skolnick’s features. The best results were obtained for monomials of degree 3. Surprisingly, for the experiments using only electric charge, the highest accuracies were obtained by monomials of degree 1. Nevertheless, results obtained for all degrees of monomials were higher than 75.8% accuracy obtained by Tilde.

6.6.2 Polynomial Features for Construction of Gene Sets

Now we describe another application of the algorithm Poly. We show how to use it to search for novel definitions of gene sets with high discriminative ability. This is useful in set-level classification methods for prediction from gene-expression data [48]. Set-level methods are based on aggregating values of gene expressions contained in pre-defined gene sets and then using these aggregated values as features for classification. Here, we, first, describe the problem and available data and then we explain how we can construct meaningful novel gene sets using Poly.

The datasets contain class-labeled gene-samples corresponding to measurements of activities of thousands of genes. Typically, the datasets contain only tens of measured samples. In addition to this *raw* measured data, we also have relational descriptions of some biological pathways from publicly available² database KEGG [54]. Each KEGG pathway is a description of some biological process (a metabolic reaction, a signalling process etc.). It contains a set of genes annotated by relational description which contains relations among genes such as *compound*, *phosphorylation*, *activation*, *expression*, *repression* etc. The relations do not necessarily refer to the

² The KEGG database is no longer available for free. However, it was still available in 2011 when these experiments were performed.

processes involving the genes per se but they may refer to relations among the products of these genes. For example, the relation *phosphorylation* between two genes A, B is used to indicate that a protein coded by the gene A adds phosphate group(s) to a protein coded by the gene B.

We constructed examples $(H_S, \vec{\theta}_S)$ from the gene-expression samples and KEGG pathways as follows. For each gene g_i , we introduced a logical atom $g(g_i, r_i)$ to capture its expression level. Then we added all relations extracted from KEGG as logical atoms $\text{relation}(g_i, g_j, \text{relationType})$. We also added a numerical indicator of class-label to each example as a logical atom $\text{label}(\pm 1)$ where +1 indicates a positive example and -1 a negative example. Finally, for each gene-expression sample S we constructed the vector of the gene-expression levels $\vec{\theta}_S$. Using the feature construction algorithm outlined in Section 6.5 we constructed a large set of tree-like features³ involving exactly one atom $\text{label}(L)$, at least one atom $g(G_i, R_i)$ and relations *expression*, *repression*, *activation*, *inhibition*, *phosphorylation*, *dephosphorylation*, *state* and *binding/association*. After that we had to select a subset of these features. Clearly, the aggregated values of meaningful gene sets should correlate with the class-label. An often used aggregation method in set-level classification methods is the *average*. Therefore what we need to do is to select features based on the correlation of the average expression of the genes assumed by the feature and the class-label but this is easy since we can get the estimate of features' covariance matrices Σ_F using polynomial features and then compute the correlation of the average expression of the assumed genes and the class-label. The absolute values of correlations give us means to heuristically order the features. Based on this ordering we found a collection of gene sets given by the features (ignoring gene sets which contained only genes contained in a union of already constructed gene sets).

Dataset	Gaussian logic	FCF
Collitis [14]	80.0	89.4
Pleural Mesothelioma [41]	94.4	92.6
Parkinson 1 [108]	52.7	54.5
Parkinson 2 [108]	66.7	63.9
Parkinson 3 [108]	62.7	77.1
Pheochromocytoma [20]	64.0	56.0
Prostate cancer [7]	85.0	80.0
Squamous cell carcinoma [60]	95.5	88.6
Testicular seminoma [40]	58.3	61.1
Wins	5	4

Table 8: Accuracies of set-level-based classifiers with Gaussian-logic features and FCF-based features, estimated by leave-one-out cross-validation.

We constructed the features using a gene-expression dataset from [34] which we did not use in the subsequent predictive classification experiments. A feature defining gene sets which exhibited one of the strongest correlations with the class-label is shown here:

$$F = \text{label}(R_1) \wedge g(A, R_2) \wedge \text{relation}(A, B, \text{phosphorylation}) \wedge \\ g(B, R_3) \wedge \text{relation}(A, C, \text{phosphorylation}) \wedge g(C, R_4)$$

We compared gene sets constructed by the outlined procedure with gene sets based on so-called *fully-coupled fluxes* (FCFs) which are biologically-motivated gene sets used previously in the context of set-level classification [48]. We constructed the same number of gene sets for our

³ We have used a subset of 50 pathways from KEGG to keep the memory consumption of the feature-construction algorithm under 1GB.

features as was the number of FCFs. The accuracies of an SVM classifier (estimated by leave-one-out cross-validation) are shown in Table 8. We can notice that the gene sets constructed using our novel method performed equally well as the gene sets based on fully-coupled fluxes. Interestingly, our gene sets contained about half the number of genes as compared to FCFs and despite that, they were able to perform equally well.

6.7 CONCLUSIONS

In this chapter, we presented an algorithm for construction of polynomial features called Poly. This algorithm is based on the algorithm HiFi from which it differs mainly in the way it filters reducible and redundant features. Poly is able to construct polynomial relational features for propositionalization which made it possible to achieve high predictive accuracies using only a limited amount of information. We also showed that it can be used for other tasks, e.g. for construction of gene sets for set-level classification.

6.8 PROOFS

Proposition 17. *Let B_1 and B_2 be pos τ_θ -blocks such that $\text{vars}(B_1) \cap \text{vars}(B_2) = \emptyset$ and let $M_1 = (B_1, (c_1, \dots, c_k))$ and $M_2 = (B_1, (d_1, \dots, d_l))$. If, for all examples $e = (H, \vec{\theta}) \in \mathcal{E}$, it holds $\text{Eval}(M_1, e) = \text{Eval}(M_2, e)$ then $N_1 = (B^- \oplus B_1, (c_1, \dots, c_k))$ and $N_2 = (B^- \oplus B_2, (d_1, \dots, d_l))$ are redundant (relatively to each other) for any neg τ_θ -block B^- .*

Proof. This follows easily from a brief inspection of Algorithm 5. A pos τ_θ -block contributes to computation of the result only through its sample parameters computed by the function Eval on line 11 of Algorithm 5. If the sample parameters of M_1 and M_2 are equal then the result of computing $\text{Eval}((B^- \oplus B_1, (c_1, \dots, c_k)), e)$ must be equal to $\text{Eval}((B^- \oplus B_2, (d_1, \dots, d_l)), e)$ for any neg τ_θ -block B^- . \square

Proposition 18. *Let B_1 and B_2 be pos τ_θ -blocks such that $\text{vars}(B_1) \cap \text{vars}(B_2) = \emptyset$. Let $F_1 = B^- \oplus B_1$ and $F_2 = B^- \oplus B_2$ be features. If, for all examples $e = (H, \vec{\theta})$ in a given dataset \mathcal{E} , it holds $\text{dom}_H(B_1) = \text{dom}_H(B_2)$ and if there is a real number c such that $\mathcal{S}(B_1\theta_{1,t}, e) = \mathcal{S}(B_2\theta_{2,t}, e)$ for all examples $e \in \mathcal{E}$ and all substitutions $\theta_{1,t} = \{p(B_1)/t\}$ and $\theta_{2,t} = \{p(B_2)/t\}$ where $t \in \text{dom}_H(B_1) = \text{dom}_H(B_2)$ then $M_{F_1} = (F_1, f)$ and $M_{F_2} = (F_2, f)$ are redundant for any function f .*

Proof. First, we can apply an observation from the proof of Proposition 5, which gives us:

$$\mathcal{S}(F_1\theta_{1,t}, e) = \mathcal{S}(B^-\theta_t, e) \times \mathcal{S}(B_1\theta_{1,t}, e)$$

where $\theta_t = \{n(B^-)/t\}$. The reason why we can do this is that $F_1\theta_{1,t}$ is disconnected and can be decomposed as $(B^-\theta_t) \wedge (B_1\theta_{1,t})$. Next, we can get completely analogically the following:

$$\begin{aligned} \mathcal{S}(F_2\theta_{2,t}, e) &= \mathcal{S}(B^-\theta_t, e) \times \mathcal{S}(B_2\theta_{2,t}, e) = \\ &= \mathcal{S}(B^-\theta_t, e) \times \mathcal{S}(B_1\theta_{1,t}, e) = \\ &= \mathcal{S}(F_1\theta_{1,t}, e) \end{aligned}$$

This, together with the fact that we can get the sample sets $\mathcal{S}(F_1, e)$ and $\mathcal{S}(F_2, e)$ by computing the unions of the above sample sets over all $t \in \text{terms}(e)$, shows that M_{F_1} and M_{F_2} are indeed redundant because their sample sets are identical and they are based on the same function f . \square

Part III

RELATIONAL LEARNING MODULO HYPOTHESIS LANGUAGES

Methods for construction of hypotheses in relational learning can be broadly classified into two large groups: methods based on specialization, so-called top-down methods, and methods based on generalization, so-called bottom-up methods. The algorithms RelF, HiFi and Poly described in Chapters 5 and 6 are examples of top-down methods. Bottom-up methods have received increased attention recently as they have been identified useful for learning in domains which require discovery of long complex rules [88, 105]. Many bottom-up relational learning algorithms are based on Plotkin’s least general generalization operator [97] which is an operator able to construct a least *general* clause which θ -subsumes a given set of clauses where *least general* is meant w.r.t. θ -subsumption order. In this chapter we describe a new generic bottom-up learning method. The key idea underlying this new algorithm is to generalize Plotkin’s least general generalization operator.

The main motivation for the work presented in this chapter was to develop methods for bottom-up relational learning that would be able to exploit structural-tractability language biases. Exploitation of structural tractability is relatively easy for top-down learning approaches but not for bottom-up ones. In top-down approaches, candidate hypotheses are usually modified by adding new literals or by unifying variables. Structural-tractability classes of clauses usually possess the so-called *hereditary property*, i.e. if a clause is from a class then any clause composed of a subset of its literals should belong to this class as well. Therefore, when searching for hypotheses from a given structural-tractability class (e.g. for treelike clauses), it suffices to discard all clauses which do not belong to it or to avoid their generation by some other means. On the other hand, in bottom-up learning approaches, one typically starts with a clause representing a learning example and tries to generalize it. Unless one restricts the possible form of learning examples, the initial clause is typically not from the given structural-tractability class and therefore it is a question how to exploit structural tractability in such bottom-up approaches. In this chapter, we introduce a generic framework able to exploit structural tractability for bottom-up learning. The framework is based on generalizing θ -subsumption, θ -reduction and least general generalization. The generalized variants of these standard operators are parametrized by sets of clauses for which these generalized operators should coincide with their standard counterparts. One of the main results presented here is a generic algorithm which is able to exploit structural tractability in bottom-up learning and which guarantees to always find a clause at least as *good* as any clause from the given class.

This chapter is organized as follows. We review necessary material regarding θ -subsumption, θ -reduction, least general generalization, constraint satisfaction, tree decompositions, treewidth and acyclicity in Section 7.1. Then we introduce a generalization of θ -subsumption in Section 7.2 and a generalization of θ -reduction in Section 7.3. Using the generalized versions of θ -subsumption and θ -reduction, we introduce so-called bounded least general generalization which is a generalized version of conventional least general generalization in Section 7.4. Then, we present a generic bottom-up learning algorithm in Section 7.5. We present several practically relevant instantiations of the introduced framework in Section 7.6 where we show that the framework can be used, for instance, with bounded-treewidth clauses, with acyclic clauses or with clauses given implicitly by constraint satisfaction local-consistency techniques. We subject the bottom-up learning method to experimental evaluation in Section 11.3. We discuss related work in Section 7.8 and conclude the chapter in Section 11.4. All proofs (with the exception of two short proofs in Section 7.1) are located in Section 7.10. In addition, we present a study of computational complexity regarding the novel concepts introduced in this chapter in Section 7.11.

7.1 PRELIMINARIES: LGG, k-CONSISTENCY, GENERALIZED ARC CONSISTENCY

An important tool exploited in this chapter, which can be used for learning clausal theories, is Plotkin's least general generalization (LGG) of clauses.

Definition 33. A clause C is said to be a least general generalization of clauses A and B (denoted by $C = \text{LGG}(A, B)$) if and only if $C \preceq_{\theta} A$, $C \preceq_{\theta} B$ and for every clause D such that $D \preceq_{\theta} A$ and $D \preceq_{\theta} B$ it holds $D \preceq_{\theta} C$.

A least general generalization of two clauses C, D can be computed in time $\mathcal{O}(|C| \cdot |D|)$. Least general generalization can be used as an operator in the process of searching for hypotheses [87, 50]. Basically, the search can be performed by iteratively applying LGG operation on examples or on already generated LGGs. A problem of approaches based on least general generalization is that the size of a LGG of a set of examples can grow exponentially in the number of examples. In order to keep the LGGs reasonably small θ -reduction is typically applied after a new LGG of some clauses is constructed [50]. Application of θ -reduction cannot guarantee that the size of LGG would grow polynomially in the worst case, however, it is able to reduce the size of the clauses (and therefore also runtime and memory consumption) significantly in non-pathological cases.

Constraint satisfaction [24] with finite domains, which was briefly reviewed in Chapter 3, can be used for solving θ -subsumption problems. A standard representation of the θ -subsumption problem in the constraint satisfaction framework was reviewed in Chapter 3. Here, we will use this representation. We have already noted that constraint satisfaction problems with treewidth bounded by k can be solved in polynomial time by the k -consistency algorithm¹. We now briefly describe the k -consistency algorithm and review some of its properties which are important for the presentation in this chapter. The description is based on the presentation by Atserias et al. [4]. Let us have a CSP $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ where \mathcal{V} is the set of variables, \mathcal{D} is the set of domains of the variables and \mathcal{C} is the set of constraints. A partial solution ϑ is an evaluation of variables from $\mathcal{V}' \subseteq \mathcal{V}$ which is a solution of the sub-problem $\mathcal{P}' = (\mathcal{V}', \mathcal{D}, \mathcal{C})$. If ϑ and φ are partial solutions, we say that φ extends ϑ (denoted by $\vartheta \subseteq \varphi$) if $\text{Supp}(\vartheta) \subseteq \text{Supp}(\varphi)$ and $V\vartheta = V\varphi$ for all $V \in \text{Supp}(\vartheta)$, where $\text{Supp}(\vartheta)$ and $\text{Supp}(\varphi)$ denote the sets of variables which are affected by the respective evaluations ϑ and φ . The k -consistency algorithm then works as follows:

k-consistency algorithm:

1. Given a constraint satisfaction problem $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ and a positive integer k .
2. Let H be the collection of all partial solutions ϑ with $|\text{Supp}(\vartheta)| < k + 1$.
3. For every $\vartheta \in H$ with $|\text{Supp}(\vartheta)| \leq k$ and every $V \in \mathcal{V}$, if there is no $\varphi \in H$ such that $\vartheta \subseteq \varphi$ and $V \in \text{Supp}(\varphi)$, remove ϑ and all its extensions from H .
4. Repeat step 3 until H is unchanged.
5. If H is empty return *false*, else return *true*.

If the k -consistency algorithm returns *true* and \mathcal{P} has treewidth bounded by k then \mathcal{P} is guaranteed to have a solution [35]. For constraint satisfaction problems with generally unbounded treewidth, k -consistency is only a necessary but not a sufficient condition to have a solution. If the k -consistency algorithm returns *false* for a CSP problem \mathcal{P} then \mathcal{P} is guaranteed to have no solutions. If it returns *true* then the problem may or may not have some solutions. So, equivalently, if the problem is soluble then k -consistency always returns *true*. This can be seen as follows. If the problem has solution then there must be a collection of partial solutions which cannot be removed by the k -consistency algorithm. This is because, for every variable V and

¹ In this chapter we follow the conventions of [4]. In other works, what we call *k-consistency* is known as *strong k + 1-consistency* [101].

every partial solution which is a subset of a complete solution of the problem, we can find another partial solution which is a superset of the partial solution and contains the variable V - simply by selecting an appropriate subset of the complete solution. Therefore the set of partial solutions reduced by the k -consistency algorithm will not become empty and the k -consistency algorithm will return *true*.

A basic property of k -consistency that we will also need is the following. If the k -consistency algorithm returns *true* for a CSP problem then it will also return *true* for any problem created from the original problem by removing some variables and some constraints, i.e. with a *subproblem*. This can be seen by noticing that if the k -consistency algorithm starts with a set H of partial solutions and returns *true* then it must also return *true* if it starts with a superset of this set. The set of partial solutions of the subproblem must necessarily be a superset of the set of partial solutions of the original problem projected on the variables of the subproblem (from monotonicity of constraints).

It is easy to check that if a clause A has treewidth bounded by k then also the CSP representation of the problem of deciding $A \preceq_{\theta} B$ has treewidth bounded by k for any clause B . It is known that due to this and due to the equivalence of CSPs and θ -subsumption, the problem of deciding θ -subsumption $A \preceq_{\theta} B$ can be solved in polynomial time when clause A has bounded treewidth (which has been known for long time by the above mentioned widely known correspondence between θ -subsumption and CSP problems).

Proposition 19. *We say that a clause A is k -consistent w.r.t. a clause B (denoted by $A \triangleleft_k B$) if and only if the k -consistency algorithm executed on the CSP representation of the problem of deciding $A \preceq_{\theta} B$ returns *true*. If A has treewidth at most k and $A \triangleleft_k B$ then $A \preceq_{\theta} B$.*

Proof. Follows directly from the solubility of CSPs with bounded treewidth by the k -consistency algorithm [4] and from the equivalence of CSPs and θ -subsumption shown earlier in this section. \square

Another class of CSPs for which efficient algorithms exist is represented by so-called acyclic CSPs. Analogically to the case of bounded treewidth of clauses, if a clause is acyclic then its CSP encoding is also acyclic. Acyclic CSPs with extensionally represented relations of constraints can be solved in time polynomial in the size of the problem by the so-called *generalized arc-consistency algorithm* [6].

Generalized arc-consistency algorithm:

1. Given a constraint satisfaction problem $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ and a positive integer k .
2. Let H_1 be the collection of all partial solutions ϑ with $|\text{Supp}(\vartheta)| = 1$.
3. Let H_2 be the collection of all partial solutions ϑ such that $\text{Supp}(\vartheta) = s$ where s is a scope of some constraint $C = (s, R) \in \mathcal{C}$.
4. Let $H = H_1 \cup H_2$
5. For every $\vartheta \in H$ with $\text{Supp}(\vartheta) = \{U\}$ and every $V \in \mathcal{V}$ such that $\{U, V\} \in s$ for some constraint $C = (s, R) \in \mathcal{C}$, if there is no $\varphi \in H$ such that $\vartheta \subseteq \varphi$ and $V \in \text{Supp}(\varphi)$, remove ϑ and all its extensions from H .
6. Repeat step 3 until H is unchanged.
7. If H is empty return *false*, else return *true*.

Note that the above pseudocode only depicts the simplest version of the algorithm, which demonstrates the main properties of generalized arc-consistency and highlights the similarities with the k -consistency algorithm. Generalized arc-consistency can be checked in time polynomial in the number of variables and constraints and in the maximum cardinality of relations of the constraints. This polynomial-time bound is useful for θ -subsumption problems $A \preceq_{\theta} B$

because the number of variables in the CSP problem is equal to the number of first-order-logic variables in the clause A , the number of constraints is equal to the number of literals in A and the maximum cardinality of relations of the constraints is bounded by the number of literals in the clause B . Therefore, generalized arc-consistency can be checked for the CSP encoding of a θ -subsumption problem in time which is polynomial in the size of the two clauses for which θ -subsumption is tested.

Importantly, generalized arc-consistency enjoys very similar useful properties as k -consistency. If the generalized arc-consistency algorithm returns false for a CSP problem then the problem is guaranteed to have no solutions. If it returns true then the problem may or may not be solvable. So similarly as for k -consistency, solubility of a CSP problem implies that the generalized arc-consistency will return true. If the generalized arc-consistency algorithm returns true for a problem then it must return true also for any of its subproblems. The justification of these useful properties would go along the same lines as the justification of the respective properties of k -consistency.

If the generalized arc-consistency algorithm returns true and the CSP problem is acyclic then the problem is also guaranteed to have a solution. So, for an acyclic clause A , the θ -subsumption problem $A \preceq_{\theta} B$ can be solved in time polynomial in the sizes of the clauses A and B by applying the generalized arc consistency algorithm on the CSP encoding of the θ -subsumption problem.

Proposition 20. *We say that a clause A is generalized arc-consistent w.r.t. a clause B (denoted by $A \triangleleft_{\text{GAC}} B$) if and only if the generalized arc-consistency algorithm executed on the CSP representation of the problem of deciding $A \preceq_{\theta} B$ returns true. If A is acyclic and $A \triangleleft_{\text{GAC}} B$ then $A \preceq_{\theta} B$.*

Proof. Follows directly from the solubility of acyclic CSPs by the generalized arc-consistency algorithm and from the equivalence of CSPs and θ -subsumption shown earlier in this section. \square

7.2 BOUNDED SUBSUMPTION

In this section, we introduce *bounded* versions of θ -subsumption and develop methods for working with them. We start by defining x -subsumption and x -equivalence which are weaker versions of θ -subsumption and θ -equivalence. The notions of x -subsumption and x -equivalence will be central tools used in this chapter.

Definition 34 (x -subsumption, x -equivalence). *Let X be a possibly infinite set of clauses. Let A, B be clauses not necessarily from X . We say that A x -subsumes B w.r.t. X (denoted by $A \preceq_X B$) if and only if $(C \preceq_{\theta} A) \Rightarrow (C \preceq_{\theta} B)$ for every clause $C \in X$. If $A \preceq_X B$ and $B \preceq_X A$ then A and B are called x -equivalent w.r.t. X (denoted by $A \approx_X B$). For a given set X , the relation \preceq_X is called x -subsumption w.r.t. X and the relation \approx_X is called x -equivalence w.r.t. X .*

When it is clear from the context, we omit the phrase *w.r.t. X* from A x -subsumes B w.r.t. X .

To illustrate the concept, we start with a really simple instance of x -subsumption – an x -subsumption w.r.t. a finite set of clauses.

Example 26. Let $X = \{D\}$ be the set containing just the clause

$$D = e(V, W) \vee e(W, X) \vee e(X, Y).$$

Let us have clauses

$$\begin{aligned} A &= e(A, B) \\ B &= e(A, B) \vee e(B, C), \end{aligned}$$

and

$$C = e(V, W) \vee e(W, X) \vee e(X, Y) \vee e(Y, Z).$$

We can check relatively easily that $A \approx_X B$, $A \preceq_X C$, $B \preceq_X C$, $C \not\preceq_X A$ and $C \not\preceq_X B$ w.r.t. the set X .

It is apparent that χ -subsumptions w.r.t. a set X containing just one clause are of little practical interest. However, the set X can also be infinite and consist, for instance, of clauses having treewidth bounded by k or having hypertreewidth bounded by l or having at most m variables etc. Such χ -subsumptions are of great practical utility as we shall see in the later sections of this chapter.

Example 27. Let us have the following two clauses:

$$\begin{aligned} C &= e(A, B) \vee e(B, C) \vee e(C, A) \\ D &= e(A, B) \vee e(B, C) \vee e(C, D) \vee e(D, A). \end{aligned}$$

For these clauses, it holds $C \preceq_X D$ and $D \preceq_X C$ w.r.t. the set of clauses with treewidth at most 1 which can be checked easily using techniques presented in Section 7.6.1. On the other hand, $C \not\preceq_X D$, $D \not\preceq_X C$ for sets of clauses with treewidth at most k where $k > 1$. This can be checked easily because C and D have treewidth 2 and therefore checking χ -subsumption $C \preceq_X D$ or $D \preceq_X C$ w.r.t. the set of clauses with treewidth $k > 1$ is equivalent to checking ordinary θ -subsumption (and clearly, $C \not\preceq_\theta D$ and $D \not\preceq_\theta C$).

Conventional θ -subsumption is a special case of χ -subsumption. It is χ -subsumption w.r.t. the set of all clauses. In order to provide more insight, we prove this formally.

Proposition 21. *θ -subsumption is χ -subsumption w.r.t. the set X of all clauses.*

The next proposition states basic properties of χ -subsumption and χ -equivalence.

Proposition 22. *Let X be a set of clauses. Then χ -subsumption w.r.t. X is a transitive and reflexive relation on clauses and χ -equivalence w.r.t. X is an equivalence relation on clauses.*

Definition 34 provides no efficient way to decide χ -subsumption between two clauses as it demands θ -subsumption of an infinite number of clauses to be tested in some cases. However, for many practically relevant sets of clauses X , there is a relation called χ -presubsumption which implies χ -subsumption and has other useful properties as we shall see later (for example, it allows quick finding of reduced versions of clauses etc.).

Definition 35 (χ -presubsumption). *Let X be a set of clauses. If \preceq_X is the χ -subsumption w.r.t. X and \triangleleft_X is a relation such that:*

1. *If $A \in X$ and $A \triangleleft_X B$ then $A \preceq_\theta B$.*
2. *If $A \preceq_\theta B$ then $A \triangleleft_X B$.*
3. *If $A \in X$, $A \triangleleft_X B$ and $B \triangleleft_X C$ then $A \triangleleft_X C$.*

then we say that \triangleleft_X is an χ -presubsumption w.r.t. the set X .

When searching for an efficient procedure for checking χ -presubsumption w.r.t. a given set X we should be looking for a procedure which is able to decide θ -subsumption problems of the type $A \preceq_\theta B$ where $A \in X$ efficiently. For example, if X is the set of all acyclic clauses and if we want a polynomial-time decidable χ -presubsumption then we should look for a procedure that is able to efficiently decide θ -subsumption problems where A is acyclic. It turns out that generalized arc-consistency is such a procedure (as we show in Section 7.6.2).

The next proposition shows that if X is a set of clauses and \triangleleft_X is an χ -presubsumption w.r.t. X then \triangleleft_X provides a sufficient condition for χ -subsumption w.r.t. X .

Proposition 23. *Let X be a set of clauses. If \preceq_X is χ -subsumption w.r.t. X and \triangleleft_X is an χ -presubsumption w.r.t. X then $(A \triangleleft_X B) \Rightarrow (A \preceq_X B)$ for any two clauses A, B (not necessarily from X).*

Note that the validity of Proposition 23 already follows from the conditions 1 and 3. The condition 2 will come into play in the next sections. Without the third condition, we could not establish bounds on the reduction procedure presented in Section 7.3.

We will use Proposition 23 in the next section where we deal with bounded reduction of clauses. We will use it for showing that certain procedures which transform clauses always produce clauses which are x -equivalent w.r.t. a given set X . We will also use it for showing that x -presubsumptions can be used for directing hypothesis search in our bottom-up learning algorithm presented in Section 7.5.

Using Definition 35 and Proposition 23, we can see that a necessary condition for a relation \triangleleft_X to be an x -presubsumption w.r.t. X is that the following holds for any clauses A and B : $(A \preceq_{\emptyset} B) \Rightarrow (A \triangleleft_X B)$ and $(A \triangleleft_X B) \Rightarrow (A \preceq_X B)$. The next proposition shows that this is actually not only a necessary but also a sufficient condition.

Proposition 24. *Let X be a set of clauses. Let \triangleleft_X be a relation such that for any two clauses A and B : $(A \preceq_{\emptyset} B) \Rightarrow (A \triangleleft_X B)$ and $(A \triangleleft_X B) \Rightarrow (A \preceq_X B)$. Then \triangleleft_X is an x -presubsumption w.r.t. the set X .*

7.3 BOUNDED REDUCTION

Proposition 23 can be used to search for clauses which are smaller than the original clause but are still x -equivalent to it. This process, which we term x -reduction, will be an essential tool in the bottom-up relational learning algorithm presented in Section 7.5.

Definition 36 (x -reduction). *Let X be a set of clauses. We say that a clause \widehat{A} is an x -reduction of clause A if and only if $\widehat{A} \preceq_{\emptyset} A$ and $A \preceq_X \widehat{A}$ and if this does not hold for any $B \subsetneq \widehat{A}$ (i.e. if there is no $B \subsetneq \widehat{A}$ such that $B \preceq_{\emptyset} A$ and $A \preceq_X B$).*

For a given clause, there may be even smaller x -equivalent clauses than its x -reductions. There may also be multiple x -reductions differing by their lengths for a single clause. These two properties of x -reduction are demonstrated in the next two examples.

Example 28. Let $X = \{C\}$ be the set containing just the clause

$$C = e(V, W) \vee e(W, X) \vee e(X, Y) \vee e(Y, Z).$$

Let us have another clause

$$A = e(A, B) \vee e(B, C) \vee e(C, A)$$

for which we want to compute its x -reduction w.r.t. the set X . We can check relatively easily (e.g. by enumerating all subsets of literals from A) that the only x -reduction of A is A itself (up to renaming of variables). However, there is also a smaller clause x -equivalent to A and that is $A' = e(X, X)$. The x -equivalence of A and A' follows from the fact that $C \preceq_{\emptyset} A$ and $C \preceq_{\emptyset} A'$ and there is no other clause other than C in the set X . It might seem that the clauses A and A' are x -equivalent only because the set X used in this example is rather pathological but, in fact, the two clauses are also x -equivalent w.r.t. the set of all clauses with treewidth at most 1.

Considering the above example, one could wonder why we did not define x -reduction of a clause as the smallest clause x -equivalent to it. As we will see later in Section 7.4, x -reduction defined in this way would be of no use in the bottom-up learning system.

The next example demonstrates the fact that some clauses may have multiple x -reductions differing in lengths.

Example 29. Let $X = \{C\}$ be the set containing just the clause

$$C = e(V, W) \vee e(W, X) \vee e(X, Y) \vee e(Y, Z).$$

It is the same set as in the previous example. Let us have another clause

$$A = e(A, B) \vee e(B, C) \vee e(C, A) \vee e(B, H) \vee e(H, I) \vee e(I, A)$$

for which we want to compute its χ -reduction w.r.t. the set X . Graphically, the clause A can be imagined as a graph with two oriented cycles (with length 3 and 4) which share one edge. There are two χ -reductions of clause A w.r.t. the set X (up to renaming of variables)

$$A \approx_X e(A, B) \vee e(B, C) \vee e(C, A)$$

and

$$A \approx_X e(B, H) \vee e(H, I) \vee e(I, A) \vee e(A, B).$$

Similarly as in the previous example, the two χ -reductions would be χ -reductions also w.r.t. the set of clauses with treewidth at most 1.

In order to be able to compute χ -reductions, we would need to be able to decide χ -subsumption. However, we very often have only an efficient χ -presubsumption. Importantly, if there is an χ -presubsumption \triangleleft_X w.r.t. a set X decidable in polynomial time then there is a polynomial-time algorithm for computing possibly larger clauses with the same properties as χ -reductions. We call this algorithm *literal-elimination algorithm*. It works as shown in the pseudo-code below.

Literal-elimination algorithm:

1. Given a clause A which should be reduced.
2. Set $A' := A$.
3. Select a literal L from A' such that $A' \triangleleft_X A' \setminus \{L\}$. If there is no such literal, return A' and finish.
4. Set $A' := A' \setminus \{L\}$
5. Go to step 3.

The next proposition states formally the properties of the literal-elimination algorithm. It also gives a bound on the size of the reduced clause which is output of the literal-elimination algorithm. This bound is given in terms of lengths of conventional θ -reductions.

Proposition 25. *Let us have a set X and a polynomial-time decision procedure for checking \triangleleft_X which is an χ -presubsumption w.r.t. the set X . Then, given a clause A on input, the literal-elimination algorithm finishes in polynomial time and outputs a clause \hat{A} satisfying the following conditions:*

1. $\hat{A} \preceq_\theta A$ and $A \preceq_X \hat{A}$ where \preceq_X is the χ -subsumption w.r.t. the set X .
2. $|\hat{A}| \leq |\hat{A}_\theta|$ where \hat{A}_θ is θ -reduction of a subset of A 's literals with maximum length.

So, the output \hat{A} of the literal-elimination algorithm has the same properties as χ -reduction ($\hat{A} \preceq_\theta A$ and $A \preceq_X \hat{A}$) with one difference and that is that it may be non-minimal in some cases. In fact, as we show in Section 7.11, the complexity of finding a *maximally reduced clause* depends on the set X , or more precisely on the available χ -presubsumption. The problem may be NP-hard even if there is a polynomial-time decidable χ -presubsumption.

Importantly, if we have a clause A and its θ -reduction \hat{A} is contained in the set X then no matter what χ -presubsumption w.r.t. X we use, the output of the literal-elimination algorithm must be a clause A' satisfying $A' \approx_\theta \hat{A}$ and $|A'| = |\hat{A}|$ (i.e. a clause isomorphic to \hat{A}). Note that A does not have to be from the set X . This is formalized in the next proposition.

Proposition 26. *Let X be a set of clauses and A be a clause. If $A_\theta \in X$ is a θ -reduction of A then $\text{litelim}_X(A) \approx_\theta A_\theta$ and $|\text{litelim}_X(A)| = |A_\theta|$ no matter which χ -presubsumption \triangleleft_X w.r.t. X is used by the literal-elimination algorithm.*

One can come up with different versions of the literal-elimination algorithm which provide different guarantees and also run in polynomial time. Nevertheless, we use the basic version described here in the experiments because, as it turns out, it is very fast in practice and reduces clauses as much as θ -reduction in most cases. One possible modification of the literal-elimination algorithm is to try to remove not individual literals, but individual variables (and all literals containing them). Such a modification of the literal-elimination algorithm has similar performance guarantees as the basic literal-elimination algorithm. It guarantees to find a clause which is at most as big as a maximum-size θ -reduction of a clause containing a subset of the variables of the original clause.

Sometimes, we may have efficient tests for χ -presubsumptions w.r.t. some sets X and Y and we might like to reduce a given clause w.r.t. the set $X \cap Y$. The next proposition shows how reduced clauses w.r.t. $X \cap Y$ can be computed using literal-elimination algorithms w.r.t. the sets X and Y .

Proposition 27. *Let X and Y be sets of clauses. Then, given a clause A , the clause computed as $\hat{A} = \text{litem}_X(\text{litem}_Y(A))$ satisfies the following conditions:*

1. $\hat{A} \preceq_{\theta} A$, $A \preceq_{X \cap Y} \hat{A}$ where $\preceq_{X \cap Y}$ is χ -subsumption w.r.t. the set $X \cap Y$.
2. $|\hat{A}| \leq |\hat{A}_{\theta}|$ where \hat{A}_{θ} is θ -reduction of a subset of A 's literals with maximum length.

7.4 BOUNDED LEAST GENERAL GENERALIZATION

In this section, we show how χ -reductions in general, and the literal-elimination algorithm in particular, can be used in bottom-up approaches to relational learning. We introduce a novel concept which we term *bounded least general generalization*. This new concept generalizes Plotkin's *least general generalization* of clauses [97].

Definition 37 (Bounded Least General Generalization). *Let X be a set of clauses. A clause B is said to be a bounded least general generalization w.r.t. the set X of clauses A_1, A_2, \dots, A_n (denoted by $B = \text{LGG}_X(A_1, A_2, \dots, A_n)$) if and only if $B \preceq_{\theta} A_i$ for all $i \in \{1, 2, \dots, n\}$ and if for every other clause $C \in X$ such that $C \preceq_{\theta} A_i$ for all $i \in \{1, 2, \dots, n\}$, it holds $C \preceq_{\theta} B$.*

Note that neither the clauses A_1, A_2, \dots, A_n nor the resulting bounded least general generalization have to be from the set X . The set X serves only to specify the clauses which, if they θ -subsume the clauses A_1, A_2, \dots, A_n , must be more general than the resulting bounded least general generalization.

The set of all bounded least general generalizations of clauses A_1, A_2, \dots, A_n w.r.t. a set X is a superset of the set of conventional least general generalizations of these clauses. This set of all bounded least general generalizations of clauses A_1, A_2, \dots, A_n is also a subset of the set of all clauses which θ -subsume all A_1, A_2, \dots, A_n . The relationship between bounded and conventional least general generalization is depicted in Fig. 8. There are two main advantages of bounded least general generalization over the conventional least general generalization. The first main advantage is that the reduced form of bounded least general generalization can be computed in polynomial time for many practically interesting sets X as we shall see later in this chapter. The second main advantage is that this reduced form can actually be smaller than the reduced form of conventional least general generalization.

The next example shows a simple illustration of bounded least general generalization.

Example 30. Let X be the set of all clauses that can be created using only literals based on predicate $a/2$. Let us have two clauses A and B for which we want to compute their bounded least general generalization w.r.t. the set X :

$$\begin{aligned} A &= a(A, B) \vee b(B, A) \\ B &= a(A, B) \vee b(B, C) \vee a(C, D) \vee b(D, A). \end{aligned}$$

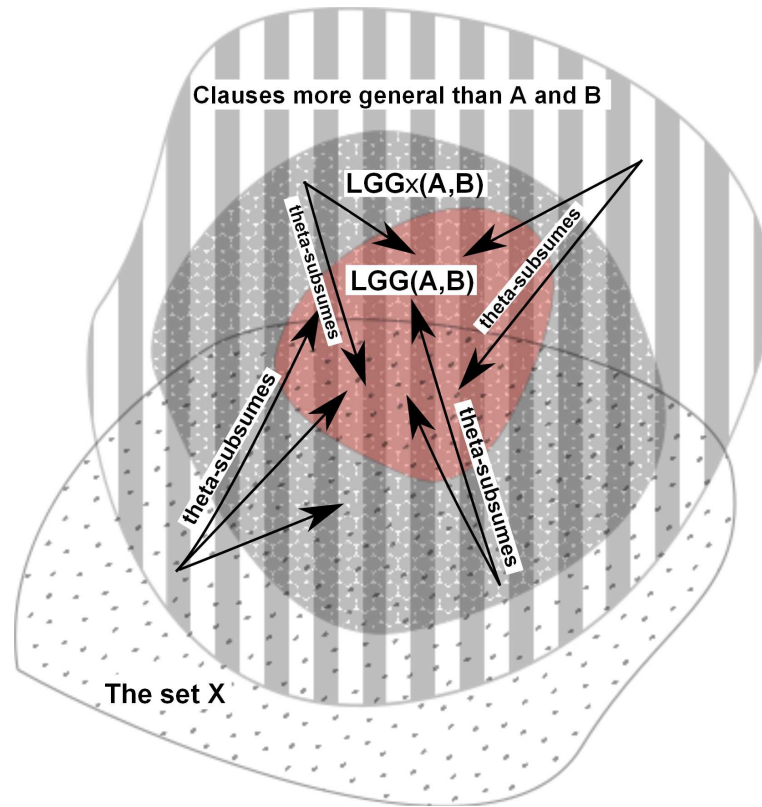


Figure 8: The relationship between the conventional least general generalization and the bounded least general generalization w.r.t. a set X .

A least general generalization of these two clauses is

$$\text{LGG}(A, B) = B = a(A, B) \vee b(B, C) \vee a(C, D) \vee b(D, A)$$

This clause is the smallest among all least general generalizations of A and B (it is equal to its own θ -reduction). A bounded least general generalization w.r.t. the set X is

$$\text{LGG}_X(A, B) = a(A, B).$$

It is instructive to see why we defined bounded least general generalization in this particular way and not in some other, seemingly more meaningful, way. Recall that our main motivation in this chapter is to be able to learn clauses more efficiently when we know (or assume) that there exist solutions to the learning problem (clauses) from some fixed potentially infinite set. Having this motivation in mind, one could argue that, for example, a more meaningful definition of bounded least general generalization should require the resulting clause to be from the set X . However, least general generalization would not exist in many cases if defined in this way, which is demonstrated in the next example.

Example 31. Let $X = \{C_1, C_2, \dots\}$ be a set of clauses of the following form:

$$\begin{aligned} C_1 &= e(A_1, A_2) \\ C_2 &= e(A_1, A_2) \vee e(A_2, A_3) \\ C_3 &= e(A_1, A_2) \vee e(A_2, A_3) \vee e(A_3, A_4) \\ &\dots \end{aligned}$$

Let us also have the following two clauses:

$$\begin{aligned} A &= e(A, B) \vee e(B, A) \\ B &= e(A, B) \vee e(B, C) \vee e(C, A) \end{aligned}$$

We would like to find a clause from X which would be their least general generalization but this is impossible for the following reason. Any clause from X θ -subsumes both A and B but none of them is least general because for any $C_i \in X$ we have $C_{i+1} \not\leq_{\theta} C_i$, $C_{i+1} \leq_{\theta} A$ and $C_{i+1} \leq_{\theta} B$. On the other hand, bounded least general generalization, as actually defined, always exists which follows trivially from the fact that the conventional least general generalization as computed by Plotkin's algorithm is also a bounded least general generalization.

Note that, as in most of the examples presented throughout this chapter, we would have to face the same problems even if X consisted of more general clauses, for example, if X consisted of clauses of treewidth bounded by some k or of acyclic clauses etc, so they are not caused by some specificities of the rather artificial set X .

We have already mentioned that reduced forms of bounded least general generalizations can often be computed in polynomial time using the literal-elimination algorithm. The method to accomplish this is based on application of χ -reductions. This is formalized in the next proposition.

Proposition 28. *Let X be a set of clauses and let \triangleleft_X be an χ -presubsumption w.r.t. the set X then the clause*

$$B_n = \text{litem}_X(\text{LGG}(A_n, \text{litem}_X(\text{LGG}(A_{n-1}, \text{litem}_X(\text{LGG}(A_{n-2}, \dots))))))$$

is a bounded least general generalization of clauses A_1, A_2, \dots, A_n w.r.t. the set X (here, $\text{litem}_X(\dots)$ denotes calls of the literal-elimination algorithm using \triangleleft_X).

Here, we can also finally see why we could not define χ -reduction of a clause C as the minimal clause χ -equivalent to it. If we defined it that way, it would no longer have to be true

$$\text{litem}_X(\text{LGG}(A_n, \text{litem}_X(\text{LGG}(A_{n-1}, \text{litem}_X(\text{LGG}(A_{n-2}, \dots)))))) \leq_{\theta} A_i$$

for all $i \in \{1, 2, \dots, n\}$ as the next example demonstrates.

Example 32. Let X be the set containing only the clause

$$C = e(A, B) \vee e(B, C) \vee e(C, D)$$

Next, let us have the following two clauses

$$\begin{aligned} A_1 &= e(A, B) \vee e(B, A) \\ A_2 &= e(A, B) \vee e(B, C) \vee e(C, A) \end{aligned}$$

Least general generalization of A_1 and A_2 is

$$B = e(A, B) \vee e(B, C) \vee e(C, D) \vee e(D, E) \vee e(E, F) \vee e(F, A)$$

The smallest clause χ -equivalent to B w.r.t. X is

$$D = e(A, A)$$

but D does not θ -subsume the original generalized clauses A_1 and A_2 . So we see that taking a clause χ -equivalent to LGG of clauses is of no use in trying to generalize them because the result may even be a clause more specific as the original clauses. Indeed, in this case, we had $D \not\leq_{\theta} A_1$, $D \not\leq_{\theta} A_2$ but $A_1 \leq_{\theta} D$ and $A_2 \leq_{\theta} D$.

On the other hand, *all* bounded least general generalizations w.r.t. the set X satisfy the conditions on generalizations, for instance,

$$E = C = e(A, B) \vee e(B, C) \vee e(C, D)$$

is a bounded least general generalization and it θ -subsumes both of the clauses A_1 and A_2 and for any clause Y from the set X which also covers them it holds $Y \leq_{\theta} E$ (here, it is actually because there is just one clause in X which is equal to E).

7.5 LEARNING WITH BOUNDED LEAST GENERAL GENERALIZATION

Conventional least general generalization can be used as an operator in searching for hypotheses [87, 50]. Basically, the search can be performed by iteratively applying LGG operation on examples or on already generated reduced LGGs. Application of reduction procedures, which compute smaller but equivalent least general generalizations, is usually necessary for keeping sizes of clauses constructed by LGG operator reasonably small. θ -reduction cannot guarantee that the sizes of least general generalizations would not grow exponentially in the worst case, but it is able to reduce the sizes of the clauses (and therefore also runtime and memory consumption) significantly in non-pathological cases. A problem of approaches based on conventional least general generalization is that computing θ -reduction is an NP-hard problem, which is especially problematic when the size of least general generalizations is large. In general, bounded reduction also cannot guarantee that the size of the constructed clauses will not be too large (in the end, θ -reduction is a special type of bounded reduction). Nevertheless, if we have a set of clauses X w.r.t. which there exists a polynomial-time literal-elimination algorithm then, at least time complexity of the reduction algorithms is not an issue. For these reasons, bounded least general generalization seems to be a suitable method for bottom-up learning.

In the simplest setting of learning single-clause theories, conventional least general generalization can be used to find a clause, if it exists, which separates positive and negative examples as:

$$H = \text{reduce}_\theta(\text{LGG}(E_n^+, \text{reduce}_\theta(\text{LGG}(E_{n-1}^+, \text{reduce}_\theta(\text{LGG}(E_{n-2}^+, \dots))))))$$

where E_i^+ are the positive examples and $\text{reduce}_\theta(\dots)$ denotes calls to the θ -reduction algorithm. One does not need negative examples in this simple case. If some clause H^* which correctly splits the given sets of positive and negative examples exists then H must also split them because it θ -subsumes all positive examples (by definition of LGG) and, if it θ -subsumed a negative example then H^* would also have to θ -subsume this example (again by definition of LGG) which would be a contradiction with H^* splitting the positive and negative examples correctly.

Importantly, one can proceed similarly when learning with bounded least general generalization. If we want to find a clause H separating positive and negative examples, we can try to construct it as:

$$H = \text{litem}_X(\text{LGG}(E_n^+, \text{litem}_X(\text{LGG}(E_{n-1}^+, \text{litem}_X(\text{LGG}(E_{n-2}^+, \dots))))))$$

In this case the conditions under which the method is guaranteed to find a solution are little different. It no longer suffices that there is some clause which correctly splits the positive and negative examples. If X is the set of clauses w.r.t. which the bounded least general generalization is computed and if there is some clause $H^* \in X$ which splits the positive and negative examples correctly then H is guaranteed to split these examples correctly. So, for example, if we believe that there exists a solution, which is an acyclic clause, then we can use bounded least general generalization w.r.t. the set of all acyclic² clauses to find a solution. If our belief is not justified since there is no solution which is an acyclic clause, then the clause H still may but also may not be a solution of the problem. The next proposition formalizes this.

Proposition 29. *Let X be a set of clauses. Let A and B be sets of clauses. If there is a clause $H^* \in X$ which θ -subsumes all clauses from A and no clauses from B then it is possible to find a clause H which also splits the two sets of clauses as:*

$$H = \text{litem}_X(\text{LGG}(A_n, \text{litem}_X(\text{LGG}(A_{n-1}, \text{litem}_X(\text{LGG}(A_{n-2}, \dots))))))$$

(here, $\text{litem}_X(\dots)$ denotes calls of the literal-elimination algorithm w.r.t. the set X).

It is interesting to note that neither the clauses which are reduced by the literal-elimination algorithm during learning nor the final constructed clause must be from the set X . It merely

² See Section 7.6.2 for a description of fast x -presubsumption algorithms w.r.t. the set of acyclic clauses.

suffices that some clause from the set X which solves the learning problem exists for the outlined method to work correctly. For example, if X is the set of all acyclic clauses, then the final learned clause can be cyclic etc.

Proposition 29 shows that if we have two sets A and B of clauses which can be separated by a clause H^* from a given set X then these sets can be also split by bounded least general generalization of clauses from one of these sets. So, for instance, if we wanted to find a single clause which correctly splits given sets of positive and negative examples under the assumption that they can be split by some clause from the set X , we could use algorithms for computing bounded least general generalizations and find the desired clause using them.

There is only a minority of problems which can be solved by finding a single clause cleanly splitting positive and negative examples. More often, it is the case that we need to find a set of clauses which together cover as many positive examples and as few negative examples as possible (which is typically expressed through maximization of a scoring function). The existing systems such as Progol [86] or ProGolem [88] usually tackle this task by an iterative covering approach in which a single clause obtaining good score is found in each iteration and the positive examples covered by it are removed from the dataset so that the clauses found in the subsequent iterations would cover the other, not yet covered, positive examples. In this section we describe an iterative covering approach using bounded least general generalization called BULL and prove its most important properties. For example, we show that the single-clause learning component always finds a clause which is at least as *good* as any clause from the set X .

We start by describing the single-clause learning component of the main algorithm called S-BULL³ (Algorithm 6). The algorithm is based on best-first search [103] in which each new candidate clause is constructed by computing bounded least general generalization of an already constructed clause with an example not yet α -subsumed by it. This way, the algorithm would be exactly the best-first search algorithm where the used scoring function is the difference of the number of covered positive and negative examples. However, unlike straightforward implementation of the best-first-search algorithm, S-BULL contains also a step in which bounded least general generalization of the newly constructed clause and the positive examples covered by it w.r.t. the α -presubsumption relation \triangleleft_X is computed (the inner-most repeat-until loop starting on line 15). This step ensures that the positive examples covered by a constructed hypothesis w.r.t. the α -presubsumption \triangleleft_X will be covered by it also w.r.t. the ordinary θ -subsumption. This step is also essential for proving the following proposition which states that the clause found by S-BULL will always be at least as good as any clause from the set X (w.r.t. the scoring function $|\text{PositiveExamplesCovered}| - |\text{NegativeExamplesCovered}|$).

Proposition 30. *Let X be a set of clauses and let \mathcal{E}^+ and \mathcal{E}^- be sets of positive and negative examples, respectively. Next, let the procedure $\text{CandidateExamples}(H, \mathcal{E}^+)$ (used by the algorithm S-BULL) always return the set \mathcal{E}^+ . If there is a clause $H \in X$ which covers $p > 1$ positive examples and n negative examples then the algorithm S-BULL always finds a clause which covers p' positive examples and n' negative examples and it holds $p' - n' \geq p - n$.*

One could wonder whether a similar strategy, searching through clauses not necessarily from the set X but using only an α -presubsumption w.r.t. the set X , could also work in a top-down approach. At least a straightforward application of this approach would not work for the following reason. Let us assume that the algorithm would start with the empty clause. It would extend clauses by application of refinement steps in which new literals would be added and perhaps some variables would be unified. Let us assume that the algorithm would only use the α -presubsumption relation for computing *quality* of the generated clauses according to which it would select the best clauses. Then it would have to face the problem how to check that the positive examples which are covered w.r.t. the α -presubsumption relation are also covered w.r.t. the ordinary θ -subsumption. This is done in the S-BULL algorithm by always replacing a candidate clause by a bounded least general generalization w.r.t. X of the clauses covered by the candidate

³ Single-Clause Bottom-Up Learner (L)

Algorithm 6 S-BULL: the clause-learning component of the main algorithm

```

1: Input: Set of positive examples  $\mathcal{E}^+$ , set of negative examples  $\mathcal{E}^-$ , seed example  $S$ .
2:  $Open := ()$  /*  $Open$  is a list with elements sorted by score. */
3:  $Closed := \{\}$  /*  $Closed$  is a set. */
4:  $PosExCovBySeed := \{E \in \mathcal{E}^+ | S \triangleleft_X E\}$ 
5:  $NegExCovBySeed := \{E \in \mathcal{E}^- | S \triangleleft_X E\}$ 
6:  $BestScore := |PosExCovBySeed| - |NegExCovBySeed|$ 
7:  $BestClause := Seed$ 
8: Store the triple  $(S, PosExCovBySeed, NegExCovBySeed)$  in the list  $Open$ .
9: Store the pair  $(PosExCovBySeed, NegExCovBySeed)$  in the set  $Closed$ .
10: while  $Open \neq \emptyset$  do
11:    $(H, \{E_{i_1}^+, E_{i_2}^+, \dots, E_{i_n}^+\}, \{E_{j_1}^-, E_{j_2}^-, \dots, E_{j_m}^-\}) :=$  get the item with the highest score from  $Open$ 
    and remove it from  $Open$ .
12:   for all  $E^* \in CandidateExamples(H, \mathcal{E}^+)$  do
13:      $H^* := LGG_X(H, E^*)$ 
14:      $PosCovered := \{E_{i_1}^+, E_{i_2}^+, \dots, E_{i_n}^+\} \cup \{E^+\}$ 
15:     repeat
16:        $NewPosCovered := \{E \in (\mathcal{E}^+ \setminus PosCovered) | H^* \triangleleft_X E\}$ 
17:        $PosCovered := PosCovered \cup NewPosCovered$ 
18:        $H^* := LGG_X(H^*, A_1, \dots, A_k)$  where  $A_i \in NewPosCovered$ 
19:     until  $NewPosCovered = \emptyset$ 
20:      $NewNegCovered := \{E \in (\mathcal{E}^- \setminus \{E_{j_1}^-, \dots, E_{j_m}^-\}) | H^* \triangleleft_X E\}$ 
21:      $NegCovered := NewNegCovered \cup \{E_{j_1}^-, \dots, E_{j_m}^-\}$ 
22:     if  $(PosCovered, NegCovered \cup NewNegCovered) \notin Closed$  then
23:       Store the triple  $(S, PosCovered, NegCovered)$  in the list  $Open$ .
24:       Store the pair  $(PosCovered, NegCovered)$  in the set  $Closed$ .
25:        $Score := |PosCovered| - |NegCovered|$ 
26:       if  $Score > BestScore$  then
27:          $BestClause := H^*$ 
28:          $BestScore := Score$ 
29:       end if
30:     end if
31:   end for
32: end while

```

clause w.r.t. the χ -presubsumption relation. However, it seems impossible to achieve this within a top-down approach (that would not use any form of least general generalizations). Such a top-down approach might often end up with clauses not covering many positive examples.

Importantly, if S-BULL is used with a polynomial-time χ -presubsumption, its runtime can be upper-bounded by a singly-exponential function whereas the worst-case complexity of similar bottom-up learning algorithms based on LGG and θ -subsumption can be upper-bounded only by a doubly-exponential function. In general, the worst-case time-complexity of S-BULL is $\mathcal{O}(2^{n^+} \cdot (n^+ + n^-) \cdot f(m^{n^+}))$ where n^+ is the number of positive learning examples, n^- is the number of negative learning examples, m is the number of literals in the biggest learning example and f is an upper-bound on the runtime of the χ -presubsumption. The term 2^{n^+} is due to the fact that bounded LGGs of all subsets of positive examples might need to be searched and the term $(n^+ + n^-) \cdot f(m^{n^+})$ represents evaluation of LGGs on the dataset. If f is a polynomial then the bound is only singly-exponential. On the other hand, if an algorithm searches for a clause maximizing a given scoring function using conventional LGG and scores the intermediate clauses using standard θ -subsumption algorithms [82], then its worst-case time complexity can be upper-bounded only by $\mathcal{O}(2^{n^+} \cdot (n^+ + n^-) \cdot m^{m^{n^+}})$. This is because LGG of n clauses can have as many literals as the product of the numbers of literals in these clauses and because deciding θ -subsumption problems of the type $A \preceq_{\theta} B$ using standard algorithms takes time $|B|^{|A|}$.

The S-BULL procedure is used as a part of the iterative covering algorithm BULL which works similarly as Golem [87] or the iterative variant of ProGolem [88]. The covering algorithm starts with the full set of positive examples. It randomly picks a positive example and uses the S-BULL procedure with the randomly picked example as a seed example to find a hypothesis (a clause) maximizing the difference of covered positive and negative examples. Then it stores the found clause in a list of already found clauses and removes the set of examples covered by the found clause from the set of examples. It then repeats this process using the remaining set of positive examples until all positive examples are removed. The result is a set of clauses.

7.6 INSTANTIATIONS OF THE FRAMEWORK

In this section, we describe four practically relevant instantiations of the framework. We show that bounded least general generalizations can be computed efficiently w.r.t. classes of clauses with bounded treewidth, w.r.t. acyclic clauses, w.r.t. treelike clauses and w.r.t. to clauses complying with a simple language bias. Finally, we also show that many CSP filtering procedures, such as path-consistency, singleton-arc-consistency etc. can be used to define other practically relevant χ -presubsumptions although the explicit nature of the sets X does not have to be known in these cases.

7.6.1 Clauses of Bounded Treewidth

One of the classes of clauses w.r.t. which the reduced forms of bounded LGGs can be computed efficiently is the class of clauses with bounded treewidth. The notion of treewidth of clauses was explained in Section 7.1. What we need to show in order to demonstrate that the framework described in this chapter can be efficiently applied in the case of bounded-treewidth clauses is that there is a polynomial-time decidable χ -presubsumption relation. In the next proposition, we show that k -consistency algorithm [4] can be used to obtain such an χ -presubsumption.

Proposition 31. *Let $k \in \mathbb{N}$ and let \triangleleft_k be a relation on clauses defined as follows: $A \triangleleft_k B$ if and only if the k -consistency algorithm run on the CSP-encoding (described in Section 7.1) of the θ -subsumption problem $A \preceq_{\theta} B$ returns true. The relation \triangleleft_k is an χ -presubsumption w.r.t. the set X_k of all clauses with treewidth at most k .*

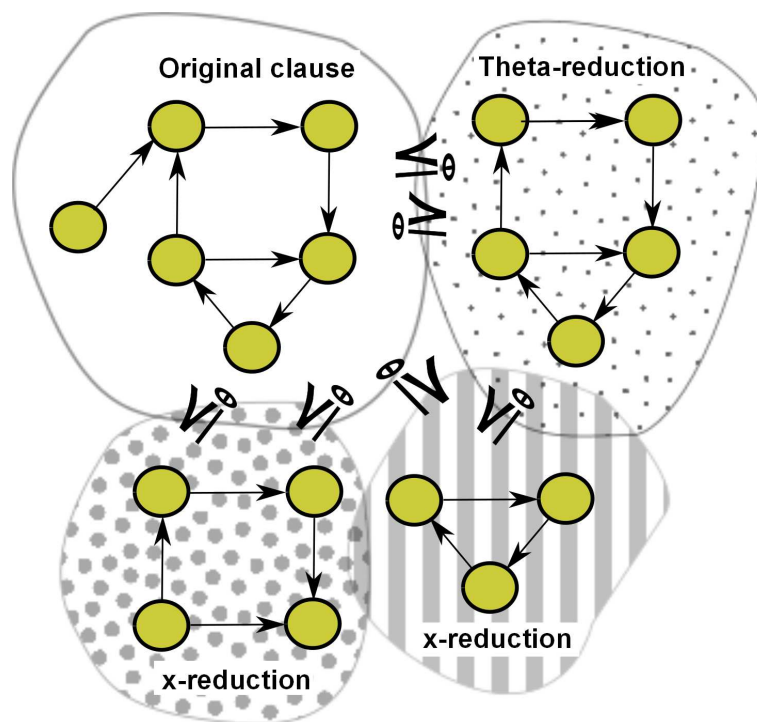


Figure 9: Illustration of θ -reduction, reduction by *literal-elimination algorithm* and reduction by *literal-substitution algorithm*.

Example 33. Let us have the following four clauses which are all mutually x -equivalent.

$$\begin{aligned}
 A &= e(A, B) \vee e(B, C) \vee e(C, E) \vee e(D, B) \vee e(D, E) \vee e(E, F) \vee e(F, D) \\
 B &= e(B, C) \vee e(C, E) \vee e(D, B) \vee e(D, E) \vee e(E, F) \vee e(F, D) \\
 C &= e(B, C) \vee e(C, E) \vee e(D, B) \vee e(D, E) \\
 D &= e(D, E) \vee e(E, F) \vee e(F, D)
 \end{aligned}$$

These clauses are depicted graphically as graphs in Figure 9 together with the θ -subsumption relations among them. The clause B is a θ -reduction of the clause A. The clauses C and D are x -reductions of A (and consequently also of B).

Low-treewidth clauses can lead to highly accurate classifiers. In our previous studies partially described in Chapter 5 and in the papers [71, 62], we observed that all clauses learned by the ILP systems Progol, nFOIL [74] and kFOIL [73] in all the conducted experiments had treewidth 1 (after the removal of the variable formally identifying the learning example) although this had not been stipulated by the language bias. In a similar spirit, Horváth and Ramon [49] note that more than 99.9 percent of molecules in the NCI repository⁴ have treewidth lower than four. The classical ILP systems produce low-treewidth clauses mostly because their top-down search strategy does not allow them to reach long clauses of higher treewidth. Learning that produces clauses as good as those with treewidth bounded by some number is therefore of considerable practical importance.

7.6.2 Acyclic and Treelike Clauses

Another class of clauses for which the framework of bounded least general generalization can be used is the class of acyclic clauses. In this case, the generalized-arc-consistency plays the role of x -presubsumption w.r.t. the set of all acyclic clauses which is justified by the following

⁴ <http://cactus.nci.nih.gov>

proposition which has a proof almost identical to the proof of the proposition stating that k -consistency is an χ -presubsumption w.r.t. the set of all clauses of treewidth at most k .

Proposition 32. *Let $\triangleleft_{\text{GAC}}$ be a relation on clauses defined as follows: $A \triangleleft_{\text{GAC}} B$ if and only if the generalized arc-consistency algorithm run on the CSP-encoding (described in Section 7.1) of the θ -subsumption problem $A \preceq_{\theta} B$ returns true. The relation $\triangleleft_{\text{GAC}}$ is an χ -presubsumption w.r.t. the set X_A of all acyclic clauses.*

A more restricted property than acyclicity is *treelikeness*. For precise characterization, we define treelikeness of hypergraphs and clauses in a manner analogical to how we defined acyclicity (Definition 12).

Definition 38 (Treelike clause). *A hypergraph (clause) C is treelike if the iteration of the following rules on C produces the empty hypergraph (conjunction):*

1. Remove a hyperedge (literal) which contains fewer than 2 vertices (variables).
2. Remove a vertex (variable) which is contained in at most one hyperedge (literal).

Since every treelike clause is acyclic, generalized arc-consistency can be used as an χ -presubsumption w.r.t. the set of treelike clauses. One could therefore wonder why we mention treelike clauses at all if their tractability is already established by the fact that they are acyclic. There are two reasons. First, treelike clauses were successfully used in several predictive experiments achieving favourable predictive accuracies when compared to hypotheses induced by standard relational learning algorithms [62] so it is worth to mention that the same class of clauses can be searched also using the framework of bounded least general generalizations. Second, and more important, another χ -presubsumption w.r.t. the set of treelike clauses can be obtained using a different CSP encoding with only binary constraints. As a consequence, χ -presubsumption w.r.t. treelike clauses can be implemented more easily and the practical implementation may actually be faster than a similar implementation of χ -presubsumption for acyclic clauses based on generalized arc-consistency.

The CSP encoding of the θ -subsumption problem which can be used to obtain an χ -presubsumption w.r.t. the set of treelike clauses can be constructed as follows. Let us have a θ -subsumption problem $A \preceq_{\theta} B$. We create a CSP variable V_l for every literal $l \in A$ and a CSP variable V_V for every first-order-logic variable $V \in \text{vars}(A)$. If V_l is a variable corresponding to a literal l , we set its domain to contain exactly the literals $l' \in B$ such that $l \preceq_{\theta} l'$ (which is checkable in linear time). If V_V is a variable corresponding to a first-order-logic variable V , then we set its domain to contain all terms in $\text{terms}(B)$. The set of constraints \mathcal{C} contains one constraint $C_{l,i}$ for every appearance of a variable in an i -th argument of a literal l . The scope of the constraint $C_{l,i}$ is set to $(V_l, V_{V_{l,i}})$ where $V_{l,i}$ is the first-order-logic variable contained in the i -th argument of the literal l . The relation of the constraint $C_{l,i}$ contains all tuples (l', t) where $l' \in B$ is a literal and t is the term in the i -th argument of l' .

Example 34. Let us have clauses A and B as follows

$$\begin{aligned} A &= \text{hasCar}(C) \vee \text{hasLoad}(C, L) \vee \text{shape}(L, \text{box}) \\ B &= \text{hasCar}(c) \vee \text{hasLoad}(c, l_1) \vee \text{hasLoad}(c, l_2) \vee \text{shape}(l_2, \text{box}) \end{aligned}$$

We now show how we can convert the problem of deciding $A \preceq_{\theta} B$ to a CSP problem using the alternative CSP encoding. Let

$$\mathcal{V} = \{V_C, V_L, V_{\text{hasCar}(C)}, V_{\text{hasLoad}(C,L)}, V_{\text{shape}(L,\text{box})}\}$$

be a set of CSP-variables. Next, let

$$\mathcal{D} = \{D_C, D_L, D_{\text{hasCar}(C)}, D_{\text{hasLoad}(C,L)}, D_{\text{shape}(L,\text{box})}\}$$

be a set of domains of variables from \mathcal{V} such that

$$\begin{aligned} D_C &= \{c, l_1, l_2\} \\ D_L &= \{c, l_1, l_2\} \\ D_{\text{hasCar}(C)} &= \{\text{hasCar}(c)\} \\ D_{\text{hasLoad}(C,L)} &= \{\text{hasLoad}(c, l_1), \text{hasLoad}(c, l_2)\} \\ D_{\text{shape}(L,\text{box})} &= \{\text{shape}(l_2, \text{box})\} \end{aligned}$$

Next, let

$$\mathcal{C} = \{C_{\text{hasCar}(C),1}, C_{\text{hasLoad}(C,L),1}, C_{\text{hasLoad}(C,L),2}, C_{\text{shape}(L,\text{box}),1}\}$$

be a set of constraints with the following scopes $(V_{\text{hasCar}(C)}, V_C)$, $(V_{\text{hasLoad}(C,L)}, V_C)$, $(V_{\text{hasLoad}(C,L)}, V_L)$ and $(V_{\text{shape}(L,\text{box})}, V_L)$ and with the relations $\{(\text{hasCar}(c), c)\}$, $\{(\text{hasLoad}(c, l_1), c), (\text{hasLoad}(c, l_2), c)\}$, $\{(\text{hasLoad}(c, l_1), l_1), (\text{hasLoad}(c, l_2), l_2)\}$ and $\{(\text{shape}(l_2, \text{box}), l_2)\}$, respectively. Then the constraint satisfaction problem given by \mathcal{V} , \mathcal{D} and \mathcal{C} represents the problem of deciding $A \preceq_{\theta} B$ as it admits a solution if and only if $A \preceq_{\theta} B$ holds.

The encoding has the convenient property that it contains only binary constraints and node-consistency and ordinary (not *generalized*) arc-consistency applied on this encoding can be used as an α -presubsumption w.r.t. the set of treelike clauses. Running the combination of node-consistency and arc-consistency on problems with binary constraints is known to be equivalent to the 1-consistency algorithm.

Proposition 33. *Let \triangleleft_1 be a relation on clauses defined as follows: $A \triangleleft_1 B$ if and only if the k -consistency algorithm with $k = 1$ run on the alternative CSP-encoding of the θ -subsumption problem $A \preceq_{\theta} B$ returns true. The relation \triangleleft_1 is an α -presubsumption w.r.t. the set X_{\top} of all treelike clauses.*

7.6.3 Clauses Given Implicitly by a Filtering Algorithm

All of the α -presubsumptions described in the previous sections, the α -presubsumption w.r.t. the set of treewidth- k clauses, the α -presubsumption w.r.t. the set of acyclic clauses and the α -presubsumption w.r.t. the set of treelike clauses, are based on local consistency techniques from constraint satisfaction. There are also other local consistency techniques in CSP which might be candidates for being α -presubsumptions w.r.t. some sets of clauses, for instance, path-consistency, singleton arc-consistency or singleton path-consistency to name at least a few of them [101]. It can be shown that these procedures can be used to obtain α -presubsumptions w.r.t. some sets although these sets may be given only implicitly by a specific local consistency techniques.

Example 35. Let us show, as an illustration, that singleton generalized arc-consistency [98] is an α -presubsumption w.r.t. some superset of the set of all acyclic clauses. What we need to know about singleton generalized arc-consistency is that if a CSP problem is singleton arc-consistent then it is also arc-consistent and that if a CSP problem is soluble then it is also singleton generalized arc-consistent. It follows that singleton generalized arc-consistency is an α -presubsumption w.r.t. the set of all acyclic clauses as it satisfies all three conditions stated in the definition of α -presubsumption (Definition 35) which follows from the fact that the generalized arc-consistency satisfies them (Proposition 32). Singleton generalized arc-consistency is stronger than generalized arc-consistency [98] and therefore the set X w.r.t. which it is an α -presubsumption may be larger than the set of acyclic clauses.

7.6.4 Clauses Constrained by a User-definable Language Bias

The sets of clauses described in the previous sections w.r.t. which we found efficient α -presubsumption tests were so far given exclusively by picking clauses for which θ -subsumption is

tractable. In machine learning, we often need to introduce a bias corresponding to apriori knowledge which we have about a problem domain at hand. In this section, we introduce a simple language bias that can be combined with the framework of bounded least general generalization.

Definition 39 (Constant Language Bias). *Constant language bias is a set $\mathcal{LB} = \{(p_i/\text{arity}_i, \{a_{i_1}, \dots, a_{i_k}\})\}$ where p_i are predicate symbols, $a_i \in \mathbf{N}$ and $\{a_{i_1}, \dots, a_{i_k}\} \subseteq \{1, \dots, \text{arity}_i\}$. A literal $l = p_i(t_1, \dots, t_k)$ is said to comply with language bias \mathcal{LB} if it contains constants in all arguments a_{i_1}, \dots, a_{i_k} . A clause C is said to comply with language bias \mathcal{LB} if all its literals comply with it.*

Informally, the *constant language bias* requires certain arguments of some literals to contain constants and not variables. We will use a simpler notation for constant language bias inspired by mode declarations known from Progol [86]. So, for example, we will write $\text{atom}(x, \#)$, $\text{bond}(x, x, \#)$ for a constant language bias $\{(\text{atom}/2, \{2\}), (\text{bond}/3, \{3\})\}$ which specifies clauses describing molecules where we require that if there is an atom in a learned hypothesis we want to know its type and, similarly, if there is a bond we also want to know whether it is a *single bond*, a *double bond* etc. For instance, the clause

$$\text{atom}(X, c) \vee \text{bond}(X, Y, \text{double}) \vee \text{atom}(Y, h)$$

complies with this language bias whereas the clause

$$\text{atom}(X, c) \vee \text{bond}(X, Y, Z) \vee \text{atom}(Y, h)$$

does not comply with it because of the variable Z in the argument where a constant should appear.

Proposition 34. *Let \mathcal{LB} be a language bias and let $X_{\mathcal{LB}}$ be the set of all clauses complying with \mathcal{LB} . Let A be a clause. If $A_{\mathcal{LB}} \subseteq A$ is a clause composed of exactly the literals from A complying with \mathcal{LB} then $A_{\mathcal{LB}} \preceq_{\theta} A$ and $A \preceq_X A_{\mathcal{LB}}$ w.r.t. the set $X_{\mathcal{LB}}$.*

The language bias can be used to define a set of clauses w.r.t. which bounded least general generalizations can be computed. This is a simple corollary of Proposition 34 because any clause $C_{\mathcal{LB}}$ obtained by dropping literals which do not comply with the given language bias has the same properties as an x -reduction except that it might be non-minimal.

Proposition 35. *Let \mathcal{LB} be a language bias and let $X_{\mathcal{LB}}$ be the set of all clauses complying with \mathcal{LB} . Let A_1, A_2, \dots, A_n be clauses and let $B = \text{LGG}(A_1, A_2, \dots, A_n)$. If $B_{\mathcal{LB}}$ is a clause obtained by removing from B all literals which do not comply with \mathcal{LB} then $B_{\mathcal{LB}}$ is a bounded LGG w.r.t. the set $X_{\mathcal{LB}}$. If, in addition, X is a set of clauses then $\text{litelim}_X(B_{\mathcal{LB}})$ is a bounded LGG w.r.t. the set $X \cap X_{\mathcal{LB}}$.*

The next example demonstrates the use of the above proposition.

Example 36. Let us have a language bias $\mathcal{LB} \approx e(x, x, \#)$ and two clauses

$$\begin{aligned} A &= e(a, b, 1) \vee e(b, a, 2) \\ B &= e(c, d, 1) \vee e(d, e, 1) \vee e(e, c, 1) \end{aligned}$$

The ordinary LGG of these clauses is

$$\text{LGG}(A, B) = e(A, B, 1) \vee e(B, C, X) \vee e(C, D, 1) \vee e(D, E, X) \vee e(E, F, 1) \vee e(F, A, X)$$

The bounded LGG w.r.t. the set $X_{\mathcal{LB}}$ is much smaller

$$\text{LGG}_{X_{\mathcal{LB}}}(A, B) = e(A, B, 1)$$

7.7 EXPERIMENTS

We performed experiments with several relational datasets in order to evaluate the practical potential of the novel concept of bounded least general generalization. We describe these experiments in this section. We start by describing the implementation details of the algorithm in Section 7.7.1. Then we describe the used datasets in Section 7.7.2. We present results of experiments comparing ordinary θ -reduction and χ -reduction in Section 7.7.3. We measure both the runtime of the algorithms as well as to what extent their outputs differ. Finally, we compare BULL with relational learning systems Aleph [111], nFOIL [74] and ProGolem [88] in Section 7.7.5.

7.7.1 Implementation

We implemented a basic version of the BULL algorithm as described in Section 7.5, using χ -presubsumption w.r.t. the intersection of the set of all treelike clauses (described in Section 7.6.2) and the set of clauses restricted by the user-definable language bias (described in Section 7.6.4). We used the AC-3 arc-consistency algorithm [81] for checking χ -presubsumption. We implemented the version of the literal-elimination algorithm presented in Section 7.3 with the following improvement. If a clause consists of several independent components⁵ then the algorithm first compares the components in a pairwise manner using the χ -presubsumption and if some component χ -subsumes another component then the algorithm removes it. Before reducing the components, the algorithm sorts them by their lengths. Another difference of the implemented algorithm as compared to the algorithm described in Section 7.5 is that if the number of literals of a component encountered during the reduction is greater than a given limit (set to infinity for the experiments reported in Section 7.7.3 and to 1000 literals in the experiments reported in Section 7.7.5) then the component is replaced by a clause composed of a randomly selected subset of its literals. The implemented algorithm also allows the user to set the maximum number of hypotheses expanded in a single run of the S-BULL procedure and the number of examples that should be sampled as candidates for being used for generalizing the current hypothesis in the S-BULL component of the algorithm. In addition, it is possible to set the maximum number of negative examples covered by a learned clause. Since we are mainly interested in the practical potential of the bounded least general generalization operation and not in the performance of heuristic strategies how to select the best theory from a set of clauses, the implemented version of BULL uses only the basic iterative covering strategy described in Section 7.5 which is repeated for the given number of times (set to 3 in all the experiments).

7.7.2 Datasets

We used several datasets from various problem domains including toxicity prediction of small molecules, estimation of therapeutic potential of antimicrobial peptides and estimation of their adverse effects or searching for characterization of CAD documents. The basic numerical properties of these datasets are listed in Table 9.

The first dataset BEE contains descriptions of 85 peptides labelled according to their haemolytic activity which is a proxy-value characterizing the ability of peptides to lyse mammal cells. The ability to lyse mammal cells is undesirable for peptides designated for therapeutic use. The peptides in this dataset were compiled from [128] by Szabóová et al. [119]. The sizes of the peptides in this dataset vary from 9 to 18 amino acids.

The second dataset, called CAD, contains 96 class-labelled examples of CAD documents describing product structures [139]. The CAD dataset is interesting in that relatively long features are needed to obtain reasonable classification accuracy.

⁵ We say that a clause C consists of more than one connected component if it can be rewritten as $C = C_1 \vee C_2$ where $C_1 \neq \emptyset$, $C_2 \neq \emptyset$ and $\text{vars}(C_1) \cap \text{vars}(C_2) = \emptyset$.

Dataset	Examples	Acc. of Majority Class
BEE	55	52.7 %
CAD	96	57.2 %
CAMEL	101	50.5 %
MUTA	188	66.5 %
PTC-FM	349	59.0 %
PTC-FR	351	65.5 %
PTC-MM	336	61.6 %
PTC-MR	344	55.8 %
RANDOM	200	50.0 %

Table 9: Datasets used in the experiments.

The third dataset, called CAMEL [16], contains spatial structures of 101 peptides, which are short sequences of amino acids, labelled according to their antimicrobial activity. Peptides in the dataset CAMEL are relatively short; each peptide is a chain of 15 amino acids. The peptide structures are described using pair-wise distances among amino acids. Each amino acid also has its type. Prediction of antimicrobial activity of peptides has potentially important applications because antimicrobial peptides are considered to be viable replacements for conventional antibiotics against which many microorganisms have already acquired resistance.

The dataset MUTA is the regression-friendly part of the well-known Mutagenesis dataset [43], which consists of 188 organic molecules marked according to their mutagenicity.

The next set of datasets originates from the Predictive Toxicology Challenge [46] which consists of more than three hundreds of organic molecules marked according to their carcinogenicity for male and female mice and rats.

The dataset RANDOM contains spatial structures of 200 peptides synthesized by Fjell et al. [32] labelled according to their antimicrobial activity. Peptides in the dataset RANDOM are chains of 9 amino acids and, thus, are smaller than peptides in the dataset CAMEL.

7.7.3 Bounded Reductions

The first question that we addressed was how much χ -reductions of clauses (w.r.t. the set of all treelike clauses) differ from the respective θ -reductions and how much faster they can be computed as compared to θ -reductions. We addressed this question in experiments presented here. We implemented a θ -reduction algorithm using the CSP representation of θ -subsumption problems. The underlying CSP solver was based on backtracking search using maintaining-arc-consistency and the min-domain heuristic [101]. The θ -reduction algorithm was implemented with the same level of sophistication as the literal-elimination algorithm. Similarly as the literal-elimination algorithm, it performed elimination of connected components first and only after that it performed elimination of individual literals. We compared this θ -reduction algorithm with the literal-elimination algorithm w.r.t. the set of all treelike clauses. We performed experiments with clauses from the datasets BEE, CAD, CAMEL, MUTA, PTC-MR and RANDOM (we omitted the remaining PTC datasets as they mostly overlap with the PTC-MR dataset). The clauses for reduction were created by sampling 1000 pairs of clauses from each dataset and then computing LGGs of these pairs of clauses w.r.t. the same language bias as used in Section 7.7.5. We measured the runtime for reduction, sizes of reduced clauses and also for each clause whether its θ -reduction and the output of the literal-elimination algorithm were isomorphic. The results are displayed in Table 10.

Dataset	Literal-elimination		θ -reduction		Isomorphic
	Runtime	Average size	Runtime	Average size	
BEE	5.2	80.5	48.7	80.5	100 %
CAD	53.0	230.3	2114.7	230.3	100 %
CAMEL	11.9	162.7	406.0	162.7	100 %
MUTA	47.3	41.3	283.0	41.3	100 %
PTC-MR	60.8	13.5	171.1	13.5	100 %
RANDOM	4.0	52.17	27.6	52.18	99.7 %

Table 10: Runtime in milliseconds and average sizes, measured as number of literals, of reduced clauses for the literal-elimination and the θ -reduction algorithm.

As can be seen from the results, the literal-elimination algorithm is not only substantially faster in most cases than the θ -reduction algorithm, but it also outputs clauses which are isomorphic (i.e. identical up to renaming of variables) to θ -reductions in majority of cases. Naturally, there are examples where the literal-elimination must return a clause non-isomorphic to the respective θ -reduction (for example when the input clause is a description of two directed circles of relatively prime lengths and the set X is the set of all treelike clauses) but it is still interesting that isomorphic clauses were returned in most cases by the literal-elimination algorithm and the θ -reduction algorithm for the six real-life datasets used in this experiment. Interestingly, when we disabled the initial stage in which components are reduced in pairwise manner, the literal-elimination algorithm started to return clauses smaller than θ -reductions (and thus non-isomorphic to it) more often which might be attributed partly to sorting of components by their sizes before their pairwise reduction.

7.7.4 Comparison with a Baseline Bottom-up Learning Algorithm

The results presented in the previous section indicate that the literal-elimination algorithm w.r.t. the set of treelike clauses can compute reductions of clauses significantly faster than the ordinary θ -reduction algorithm and that the sizes of the reduced clauses are usually equal or even smaller than those of the respective θ -reductions. These results do not answer another question and that is how much faster bottom-up relational learning can be and how much of its accuracy is lost when bounded least general generalization is used instead of the ordinary least general generalization. In order to answer this question, we compared BULL using the χ -presubsumption w.r.t. the set of treelike clauses (T-BULL) and BULL using θ -subsumption as the χ -presubsumption w.r.t. the set of all clauses (θ -BULL). In θ -BULL, we disabled the loop which ensures that all clauses covered w.r.t. a chosen χ -presubsumption will be covered also w.r.t. θ -subsumption (because it is unnecessary when the χ -presubsumption is θ -subsumption). We used T-BULL and θ -BULL with the following settings for all datasets. We set the maximum number of expanded hypotheses to 30 and the maximum number of covered negative examples to 0. The results of 10-fold cross-validation are shown in Table 11. It can be seen from the results that the accuracies are almost identical but that T-BULL is substantially faster than θ -BULL – for instance, more than two orders of magnitude for the PTC-FR dataset and even more for the PTC-MR dataset where we had to stop cross-validation for θ -BULL after a month of running.

7.7.5 Comparison with Existing Relational Learning Algorithms

What is very important for judging the practical utility of bounded least general generalization is whether algorithms based on it can be competitive with existing relational learners in terms

Dataset	T-BULL		θ -BULL	
	Runtime [s]	Accuracy [%]	Runtime [s]	Accuracy [%]
BEE	47	57.0 \pm 17.5	101	57.0 \pm 17.5
CAD	1493	88.6 \pm 10.3	9923	88.6 \pm 10.3
CAMEL	462	86.2 \pm 8.3	9078	86.2 \pm 8.3
MUTA	1095	77.1 \pm 7.6	7654	75.9 \pm 8.2
PTC-FM	685	60.8 \pm 5.1	2087	61.3 \pm 4.8
PTC-FR	805	69.0 \pm 6.1	107636	67.3 \pm 6.8
PTC-MM	597	63.1 \pm 2.9	2393	62.8 \pm 4.8
PTC-MR	1194	57.3 \pm 6.3	DNF	DNF
RANDOM	179	88.0 \pm 10.1	335	88.0 \pm 10.1

Table 11: Runtime in seconds and accuracy estimated by 10-fold cross-validation for BULL using χ -presubsumption w.r.t. the set of treelike clauses (T-BULL) and for BULL using θ -subsumption (θ -BULL).

of predictive accuracy. For this reason, we performed predictive-classification experiments with the datasets described in Section 7.7.2, in which we measured predictive accuracy (estimated by 10-fold cross-validation) of BULL and relational learning systems Aleph, nFOIL and ProGolem.

We set parameters of all four systems so that their runtime would be in the same orders of magnitude (tens of minutes per fold at most). We used BULL with the following settings for all the datasets. We set the maximum number of expanded hypotheses to 30 and the maximum number of covered negative examples to 0. For Aleph, we set the number of searched nodes to 50000, the *noise* parameter to 5 % and the maximum clause length to 100. nFOIL was used with maximum clause length set to 20 for all datasets and with beam size set to 100 for BEE, CAMEL, MUTA, RANDOM and CAD and 30 for the four PTC datasets. For ProGolem, we used most of the parameters with their default values, except the clause-evaluation function, which we set to use the Subsumer algorithm [104], and the maximum number of covered negative examples. We used two different settings of the lastly mentioned parameter, giving rise to two columns in Table 12: ProGolem₁ and ProGolem₂. ProGolem₁ refers to ProGolem with the default setting for the maximum number of covered negative examples, whereas ProGolem₂ refers to ProGolem with the maximum number of covered negative examples set to 0.

Results estimated by 10-fold cross-validation are shown in Table 12. Pairwise comparison of the four relational learning systems is shown in Table 13. The results shown in Tables 12 and 13 indicate that BULL is very competitive to existing relational learning systems, including two recent ones: nFOIL and ProGolem. It is interesting that Aleph performed comparably to nFOIL in terms of *wins* against the other systems although it has been shown to perform worse in [74]. However, a closer examination of the results in Table 12 shows that most of the *wins* of Aleph over nFOIL were obtained on the PTC datasets where overfitting seems to be one of the main concerns. Thus, the apparent competitiveness of Aleph to nFOIL is given mostly by the fact that the datasets used in our experiments involve several datasets where predictive accuracy of most systems is not too far from the accuracy of the classifier voting for the majority-class.

7.8 RELATED WORK

The first method that used least general generalization for clause learning was Golem [87]. Golem was restricted to ij-determinate clauses in order to cope with the possibly exponential growth of LGGs. However, most practical relational learning problems are highly non-determinate (e.g. problems involving molecules). A different approach was taken by Muggleton

Dataset	T-BULL	Aleph	ProGolem ₁	ProGolem ₂	nFOIL
BEE	57.0 ± 17.5	53.8 ± 16.5	56.8 ± 16.3	57.5 ± 20.8	47.7 ± 18.7
CAD	88.6 ± 10.3	85.7 ± 10.7	86.3 ± 7.2	87.5 ± 8.4	96.9 ± 5.2
CAMEL	86.2 ± 8.3	72.4 ± 14.5	80.5 ± 13.1	78.3 ± 10.1	83.3 ± 11.3
MUTA	77.1 ± 7.6	60.8 ± 8.9	66.5 ± 4.5	83.2 ± 7.0	76.6 ± 9.6
PTC-FM	60.8 ± 5.1	62.0 ± 3.3	59.1 ± 7.1	61.9 ± 6.9	60.2 ± 8.7
PTC-FR	69.0 ± 6.1	68.7 ± 3.9	65.0 ± 8.3	65.8 ± 4.7	67.0 ± 6.5
PTC-MM	63.1 ± 2.9	59.8 ± 5.0	60.7 ± 8.2	59.5 ± 3.6	63.1 ± 8.8
PTC-MR	57.3 ± 6.3	59.3 ± 4.5	54.9 ± 5.4	56.7 ± 7.4	57.3 ± 7.1
RANDOM	88.0 ± 10.1	86.0 ± 6.1	88.0 ± 6.3	81.0 ± 9.4	83.0 ± 5.9

Table 12: Accuracy estimated by 10-fold cross-validation for the following systems: T-BULL, Aleph, ProGolem with default maximum number of covered negative examples (ProGolem₁), ProGolem with zero maximum number of covered negative examples and nFOIL.

	Aleph	ProGolem ₁	ProGolem ₂	nFOIL
T-BULL	7/2/0	8/0/1	6/3/0	6/1/2
Aleph	-	3/6/0	5/4/0	5/4/0
ProGolem ₁	-	-	3/6/0	2/7/0
ProGolem ₂	-	-	-	3/6/0

Table 13: Wins/losses/ties for the following systems: BULL, Aleph, ProGolem with default maximum number of covered negative examples (ProGolem₁), ProGolem with zero maximum number of covered negative examples and nFOIL.

et al. [88] who introduced an algorithm called ProGolem. ProGolem is based on so-called asymmetric relative minimal generalizations (ARMGs) of clauses relative to a bottom clause. Size of ARMGs is bounded by the size of bottom clause so there is no exponential growth of the sizes of clauses. However, ARMGs are not unique and are not *least-general*.

Recently, an approach related to ours has been introduced [78] in which arc-consistency was used for structuring the space of graphs. There, arc-consistency was used as a covering operator called AC-projection. In contrast, we do not use the weaker versions of θ -subsumption (χ -subsumptions) as covering operators in this chapter but we use them only for reduction of clauses and for guiding the search which allows us to guarantee that, for instance, if a solution of standard learning problems with bounded-treewidth exists, our method is able to solve the learning problem. Thus, our approach provides theoretical guarantees which relate directly to learning problems with standard notions of covering (i.e. θ -subsumption), whereas the other approach can provide guarantees only w.r.t to the weaker (and less intuitive) AC-projection covering relation. Our framework is also more general in that it allows various different classes of clauses w.r.t. which it can work.

Example 37. Let X be the set of clauses with treewidth 1. Let us have the following set of positive examples:

$$\begin{aligned}
 P_1 &= l(a) \vee e(a, b) \vee e(b, c) \vee e(c, a) \\
 P_2 &= l(a) \vee e(a, b) \vee e(b, c) \vee e(c, d) \vee l(d) \vee e(d, e) \vee e(e, f) \vee (f, a)
 \end{aligned}$$

and the following set of negative examples:

$$\begin{aligned} N_1 &= l(a) \vee e(a, b) \vee e(b, c) \vee e(c, d) \vee l(d) \\ N_2 &= l(a) \vee e(a, b) \vee e(b, a) \\ N_3 &= l(a) \vee e(a, b) \vee e(b, c) \vee l(c) \vee e(c, d) \vee (d, a) \end{aligned}$$

A solution that would be found by the S-BULL algorithm w.r.t. the set X is

$$H = l(A) \vee e(A, B) \vee e(B, C) \vee e(C, D) \vee l(D) \vee e(D, E) \vee e(E, F) \vee (F, A)$$

It is easy to verify that H correctly splits the positive and the negative clauses. On the other hand, the method based on arc-consistency projection might return the following clause as a solution⁶.

$$H' = l(A) \vee e(A, B) \vee e(B, C) \vee e(C, A)$$

We may notice that H' would not be a correct solution of the learning problem w.r.t. the ordinary θ -subsumption because it does not θ -subsume the positive example P_2 (the clause H' covers the positive example P_2 only w.r.t. the arc-consistency projection).

Another approach related to ours is the work of Horváth et al. [50] which is also based on application of least general generalization. Their approach relies on the fact that least general generalization of treelike clauses is again a treelike clause. Since treelike clauses can be reduced in polynomial time and since θ -subsumption problems $A \preceq_{\theta} B$ where A is treelike can be decided in polynomial time as well, it is possible to search for hypotheses in a manner similar to ours using only polynomial-time reduction and θ -subsumption. As in our approach, the size of the clauses constructed as least general generalizations may grow exponentially with the number of learning examples. However, unlike our approach, the approach of Horváth et al. requires learning examples to be treelike. Our approach is therefore more general even if we consider just bounded least general generalization w.r.t. the set of treelike clauses. If the learning examples are all treelike then our method is equivalent to the method of Horváth et al. However, if the examples are not treelike, their method cannot be used at all whereas our method still guarantees to find a hypothesis at least as good as the best treelike hypothesis while using only polynomial-time χ -subsumption and χ -reduction.

In a similar spirit, Schietgat et al. [109] introduced a new method based on computing maximum common subgraphs of outerplanar graphs under so-called *block-and-bridge-preserving isomorphism* which can be done in polynomial time. This method was demonstrated to be highly competitive to *all-inclusive* strategies based on enumeration of all frequent graphs while using much lower number of maximum common subgraphs. Since it requires learning examples to be outerplanar graphs, it could not be applied to some of our datasets (e.g. the CAD dataset or the datasets of antimicrobial peptides CAMEL, BEE and RANDOM). Aside this, another difference to our method is that it is based on a restricted form of subgraph isomorphism whereas our method is based on θ -subsumption, i.e. on homomorphism.

The work presented in this chapter also relates closely to some studies of graph homomorphisms. Specifically, it was shown by Hell et al. [45] that if k -consistency check succeeds for a pair of graphs G and H then any graph with treewidth at most k which is homomorphic to G must be also homomorphic to H . This corresponds to Proposition 31 in which we showed that k -consistency is an χ -presubsumption w.r.t. the set of clauses with treewidth at most k . On the other hand, we are not aware of any work that would demonstrate the similar property for acyclic clauses, as we did in Proposition 32, or other types of clauses that we considered in this chapter. Moreover, as their motivation was different from ours, Hell et al. also did not ponder the implications of this result for machine learning.

⁶ The solution as well as the learning examples would be represented as labelled directed graphs equivalent to the clauses used here.

7.9 CONCLUSIONS

In this chapter, we introduced the first bottom-up relational learning algorithm based on least general generalization which can exploit structural tractability bias while not restricting the form of learning examples. The algorithm is generic and can be used w.r.t. various sets X , representing the language bias. It is guaranteed to find a clause which is at least as *good* as any clause from the given set X . The returned clause itself does not have to be from the set X . If the set X contains only clauses for which θ -subsumption can be decided in polynomial time and if there is a suitable polynomial-time decidable χ -presubsumption then the algorithm can use polynomial-time procedures for computing reduced forms of least general generalizations of clauses and for computing their coverage. Although even then it may run for an exponentially long time, its worst-case time complexity is only singly-exponential in the size of input whereas the time complexity of the respective algorithm based on conventional least general generalization is doubly-exponential. Prior to this work, the only way structural tractability bias could be exploited in bottom-up learning systems was to restrict the learning examples to be from a set for which θ -subsumption could be decided efficiently and which would be closed under formation of least general generalizations. Such an approach was pursued e.g. by Horváth et al. [50] for treelike clauses. Importantly, when subjected to comparative experimental evaluation, the new algorithm turned out to be very competitive to state-of-the-art relational learning systems.

7.10 PROPOSITIONS AND PROOFS

This section provides proofs of propositions stated in the main text.

Proposition 21. *θ -subsumption is the χ -subsumption w.r.t. the set X of all clauses.*

Proof. Let X be the set of all clauses. We need to show that for any clauses A, B it holds $(A \preceq_X B) \Leftrightarrow (A \preceq_\theta B)$ w.r.t. the set X . (i) We start by showing that $(A \preceq_X B) \Rightarrow (A \preceq_\theta B)$ holds. If $(A \not\preceq_X B)$ then the implication is vacuously true, so we assume that $A \preceq_X B$. The definition of χ -subsumption tells us that if $A \preceq_X B$ then $(C \preceq_\theta A) \Rightarrow (C \preceq_\theta B)$ for any $C \in X$. If we set $C := A$ in this implication (recall that $A \in X$) we get $(A \preceq_\theta A) \Rightarrow (A \preceq_\theta B)$ from which $A \preceq_\theta B$ follows. (ii) It remains to show validity of the other direction of the implication, i.e. $(A \preceq_X B) \Leftarrow (A \preceq_\theta B)$. Again, we can assume $(A \preceq_\theta B)$ because the implication is otherwise vacuously true. We need to show that for any clause $C \in X$ it holds $(C \preceq_\theta A) \Rightarrow (C \preceq_\theta B)$ but this already follows from $A \preceq_\theta B$ and from transitivity of θ -subsumption. \square

Proposition 22. *Let X be a set of clauses. Then χ -subsumption w.r.t. X is a transitive and reflexive relation on clauses and χ -equivalence w.r.t. X is an equivalence relation on clauses.*

Proof. These properties of χ -subsumption and χ -equivalence can be shown very easily.

1. Transitivity of χ -subsumption: Let $A \preceq_X B$ and $B \preceq_X C$. We need to show that then necessarily also $A \preceq_X C$, i.e. that for any clause $D \in X$ such that $D \preceq_\theta A$ it also holds $D \preceq_\theta C$. This is straightforward because if $D \preceq_\theta A$ then $D \preceq_\theta B$ (from $A \preceq_X B$) and also $D \preceq_\theta C$ (from $B \preceq_X C$).
2. Reflexivity of χ -subsumption: obvious.
3. χ -equivalence is an equivalence relation: Reflexivity and transitivity of χ -equivalence follow from reflexivity and transitivity of χ -subsumption. It remains to show that χ -equivalence is also symmetric but that follows immediately from $(A \approx_X B) \Leftrightarrow (A \preceq_X B \wedge B \preceq_X A)$.

\square

Proposition 23. *Let X be a set of clauses. If \preceq_X is χ -subsumption w.r.t. X and \triangleleft_X is an χ -presubsumption w.r.t. X then $(A \triangleleft_X B) \Rightarrow (A \preceq_X B)$ for any two clauses A, B (not necessarily from X).*

Proof. We need to show that if $A \triangleleft_X B$ then $(C \preceq_\theta A) \Rightarrow (C \preceq_\theta B)$ for all clauses $C \in X$. First, if $A \triangleleft_X B$ and $C \not\preceq_\theta A$ then the proposition holds trivially. So we assume $C \preceq_\theta A$ which implies $C \triangleleft_X A$ (using the condition 1 from Definition 35). Since $C \in X$ and $A \triangleleft_X B$, we also get $C \triangleleft_X B$ (using the condition 3 from Definition 35) and consequently also $C \preceq_\theta B$ (using the condition 1 from Definition 35). \square

Proposition 24. *Let X be a set of clauses. Let \triangleleft_X be a relation such that for any two clauses A and B : $(A \preceq_\theta B) \Rightarrow (A \triangleleft_X B)$ and $(A \triangleleft_X B) \Rightarrow (A \preceq_X B)$. Then \triangleleft_X is an x -presubsumption w.r.t. the set X .*

Proof. We show that any relation satisfying the conditions from the proposition must also satisfy all conditions from the definition of x -presubsumption (Definition 35).

1. *If $A \in X$ and $A \triangleleft_X B$ then $A \preceq_\theta B$:* If $A \in X$ then $A \preceq_\theta B$ is equivalent to $A \preceq_X B$. Therefore this condition follows from $(A \triangleleft_X B) \Rightarrow (A \preceq_X B)$.
2. *If $A \preceq_\theta B$ then $A \triangleleft_X B$:* Obvious.
3. *If $A \in X$, $A \triangleleft_X B$ and $B \triangleleft_X C$ then $A \triangleleft_X C$:* We assume that $A \in X$, $A \triangleleft_X B$ and $B \triangleleft_X C$ (because otherwise the condition would be trivially satisfied). First, $A \triangleleft_X B$ implies $A \preceq_X B$. Since $A \in X$, we get also $A \preceq_\theta B$. Next, $B \triangleleft_X C$ gives us $B \preceq_X C$. Finally, $A \in X$, $A \preceq_\theta B$ and $B \preceq_X C$ gives us $A \preceq_\theta C$ using definition of x -presubsumption which implies $A \triangleleft_X C$. \square

Proposition 25. *Let us have a set X and a polynomial-time decision procedure for checking \triangleleft_X which is an x -presubsumption w.r.t. the set X . Then, given a clause A on input, the literal-elimination algorithm finishes in polynomial time and outputs a clause \hat{A} satisfying the following conditions:*

1. $\hat{A} \preceq_\theta A$ and $A \preceq_X \hat{A}$ where \preceq_X is an x -subsumption w.r.t. the set X .
2. $|\hat{A}| \leq |\hat{A}_\theta|$ where \hat{A}_θ is θ -reduction of a subset of A 's literals with maximum length.

Proof. We start by proving $\hat{A} \preceq_\theta A$ and $A \preceq_X \hat{A}$. This can be shown as follows. First, $A \preceq_X A'$ holds in any step of the algorithm which follows from $(A' \triangleleft_X A' \setminus \{L\}) \Rightarrow (A' \preceq_X A' \setminus \{L\})$ and from transitivity of x -subsumption. Consequently we also have $A \preceq_X \hat{A}$ because $\hat{A} = A'$ in the last step of the algorithm. Second, $\hat{A} \preceq_\theta A$ because $\hat{A} \subseteq A$. Now, we prove the second part of the proposition. What remains to be shown is that the resulting clause \hat{A} will not be bigger than \hat{A}_θ . Since $\hat{A} \subseteq A$, it suffices to show that \hat{A} cannot be θ -reducible. Let us assume, for contradiction, that it is θ -reducible. If \hat{A} was θ -reducible, there would have to be a literal $L \in \hat{A}$ such that $\hat{A} \preceq_\theta \hat{A} \setminus \{L\}$. The relation \triangleleft_X satisfies $(A \preceq_\theta B) \Rightarrow (A \triangleleft_X B)$ therefore it would also have to hold $A' \triangleleft_X A' \setminus \{L\}$. However, then L should have been removed by the literal-elimination algorithm which is a contradiction with \hat{A} being output of it. The fact that the literal-elimination algorithm finishes in polynomial time follows from the fact that, for a given clause A , it calls the polynomial-time procedure for checking the relation \triangleleft_X at most $|A|^2$ times (the other operations of the literal-elimination algorithm can be performed in polynomial time as well). \square

Proposition 26. *Let X be a set of clauses and A be a clause. If $A_\theta \in X$ is a θ -reduction of A then $\text{litelim}_X(A) \approx_\theta A_\theta$ and $|\text{litelim}_X(A)| = |A_\theta|$ no matter which x -presubsumption \triangleleft_X w.r.t. X is used by the literal-elimination algorithm.*

Proof. Since A_θ is a θ -reduction of A , there is a substitution ϑ such that $A_\theta \vartheta \subseteq A$. In what follows, we assume without loss of generality that $A_\theta \subseteq A$. First, we show that for all intermediate clauses A' which occur during the processing of the literal-elimination algorithm, it must always hold $A' \preceq_\theta A_\theta$. We have $A_\theta \preceq_\theta A$ from $A_\theta \subseteq A$. Then we can use transitivity of x -subsumption to obtain $A_\theta \preceq_X A'$ from $A_\theta \preceq_X A$ and $A \preceq_X A'$ (this always holds for any A' in the literal-elimination algorithm). The fact $A_\theta \preceq_X A'$ together with $A_\theta \in X$ can be used to infer $A_\theta \preceq_\theta A'$.

In addition, we have $A' \subseteq A$ and $A \preceq_{\theta} A_{\theta}$ from which we obtain $A' \preceq_{\theta} A_{\theta}$. Since the last A' occurring in the literal-elimination algorithm is also its output, we can combine these results to get the first part of the proposition, i.e. that $\text{litem}_X(A) \approx_{\theta} A_{\theta}$. Now, we show the second part of the proposition. It suffices to show that $\text{litem}_X(A)$ is not θ -reducible because if it is θ -equivalent to a θ -reduction of A (as we have shown) and if it is not θ -reducible, it must necessarily be a θ -reduction of A and therefore $|\text{litem}_X(A)| = |A_{\theta}|$. We can show this by contradiction in a similar way to how we proceeded in the proof of Proposition 25. We denote $\hat{A} = \text{litem}_X(A)$. Let us assume, for contradiction, that \hat{A} is θ -reducible. If it was θ -reducible, there would have to be a literal $L \in \hat{A}$ such that $\hat{A} \preceq_{\theta} \hat{A} \setminus \{L\}$. The relation \triangleleft_X satisfies $(A \preceq_{\theta} B) \Rightarrow (A \triangleleft_X B)$ therefore it would also have to hold $\hat{A} \triangleleft_X \hat{A} \setminus \{L\}$. However, then L should have been removed by the literal-elimination algorithm which is a contradiction with \hat{A} being output of it. \square

Proposition 27. *Let X and Y be sets of clauses. Then, given a clause A , the clause computed as $\hat{A} = \text{litem}_X(\text{litem}_Y(A))$ satisfies the following conditions:*

1. $\hat{A} \preceq_{\theta} A$, $A \preceq_{X \cap Y} \hat{A}$ where $\preceq_{X \cap Y}$ is x -subsumption w.r.t. the set $X \cap Y$.
2. $|\hat{A}| \leq |\hat{A}_{\theta}|$ where \hat{A}_{θ} is θ -reduction of a subset of A 's literals with maximum length.

Proof. We denote $B = \text{litem}_Y(A)$. It holds $B \preceq_{\theta} A$ and $A \preceq_Y B$ by Proposition 25 and therefore also $A \preceq_{X \cap Y} B$ (because $X \cap Y \subseteq Y$). When the literal-elimination algorithm $\text{litem}_X(\dots)$ is applied on B , giving $\hat{A} = \text{litem}_X(B)$, it must hold $\hat{A} \preceq_{\theta} B$ which implies $\hat{A} \preceq_{\theta} A$ (by transitivity). Furthermore, it must also hold $B \preceq_X \hat{A}$ which implies $B \preceq_{X \cap Y} \hat{A}$ (again because $X \cap Y \subseteq X$). So, we have $A \preceq_{X \cap Y} B$ and $B \preceq_{X \cap Y} \hat{A}$ which gives us $A \preceq_{X \cap Y} \hat{A}$. It remains to show $|\hat{A}| \leq |\hat{A}_{\theta}|$ but this already follows from the fact that $|B| \leq |\hat{A}_{\theta}|$ and by the fact that the literal-elimination never increases sizes of clauses. \square

Proposition 28. *Let X be a set of clauses and let \triangleleft_X be an x -presubsumption w.r.t. the set X then the clause*

$$B_n = \text{litem}_X(\text{LGG}(A_n, \text{litem}_X(\text{LGG}(A_{n-1}, \text{litem}_X(\text{LGG}(A_{n-2}, \dots))))))$$

is a bounded least general generalization of clauses A_1, A_2, \dots, A_n w.r.t. the set X (here, $\text{litem}_X(\dots)$ denotes calls of the literal-elimination algorithm using \triangleleft_X).

Proof. First, we show that $B \preceq_{\theta} A_i$ for all $i \in \{1, 2, \dots, n\}$ using induction on n . The base case $n = 1$ is obvious since then $B_1 = A_1$ and therefore $B_1 \preceq_{\theta} A_1$. Now, we assume that the claim holds for $n - 1$ and we will show that then it must also hold for n . First, $B_n = \text{LGG}(A_n, B_{n-1})$ θ -subsumes the clauses A_1, \dots, A_n which can be checked by recalling the induction hypothesis and definition of LGG. Second, $\text{litem}_k(\text{LGG}(A_n, B_{n-1}))$ must also θ -subsume the clauses A_1, \dots, A_n because $\text{litem}_k(\text{LGG}(A_n, B_{n-1})) \subseteq \text{LGG}(A_n, B_{n-1})$.

Again using induction, we now show that $C \preceq_{\theta} B_n$ for any $C \in X$ which θ -subsumes all A_i where $i \in \{1, \dots, n\}$. The base case $n = 1$ is obvious since then $B_1 = A_1$ and therefore every C which θ -subsumes A_1 must also θ -subsume B_1 . Now, we assume that the claim holds for $n - 1$ and we prove that it must also hold for n . That is we assume that

$$C' \preceq_{\theta} B_{n-1} = \text{litem}_X(\text{LGG}(A_{n-1}, \text{litem}_X(\text{LGG}(A_{n-2}, \text{litem}_X(\text{LGG}(A_{n-3}, \dots))))))$$

for any $C' \in X$ which θ -subsumes the clauses A_1, A_2, \dots, A_{n-1} . We show that then it must also hold $C \preceq_{\theta} B_n = \text{litem}_X(\text{LGG}(A_n, B_{n-1}))$ for any $C \in X$ which θ -subsumes the clauses A_1, A_2, \dots, A_n . We have $C \preceq_{\theta} \text{LGG}(A_n, B_{n-1})$ because $C \preceq_{\theta} B_{n-1}$ which follows from the induction hypothesis and because any clause which θ -subsumes both A_n and B_{n-1} must also θ -subsume $\text{LGG}(A_n, B_{n-1})$ (from the definition of LGG). It remains to show that C also θ -subsumes $\text{litem}_X(\text{LGG}(A_n, B_{n-1}))$. This follows from

$$\text{LGG}(A_n, B_{n-1}) \preceq_X \text{litem}_X(\text{LGG}(A_n, B_{n-1}))$$

(which is a consequence of Proposition 25) because if

$$\text{LGG}(A_n, B_{n-1}) \preceq_X \text{litelim}_X(\text{LGG}(A_n, B_{n-1}))$$

then

$$(C \preceq_\theta \text{LGG}(A_n, B_{n-1})) \Rightarrow (C \preceq_\theta \text{litelim}_X(\text{LGG}(A_n, B_{n-1})))$$

for any clause $C \in X$ (this is essentially the definition of x -subsumption). \square

Proposition 29. *Let X be a set of clauses. Let \mathcal{A} and \mathcal{B} be sets of clauses. If there is a clause $H^* \in X$ which θ -subsumes all clauses from \mathcal{A} and no clauses from \mathcal{B} (i.e. $\forall A \in \mathcal{A} : H^* \preceq_\theta A$ and $\forall B \in \mathcal{B} : H^* \not\preceq_\theta B$) then it is possible to find a clause H which splits the two sets of clauses in the same way as:*

$$H = \text{litelim}_X(\text{LGG}(A_n, \text{litelim}_X(\text{LGG}(A_{n-1}, \text{litelim}_X(\text{LGG}(A_{n-2}, \dots))))))$$

(here, $\text{litelim}_X(\dots)$ denotes calls of the literal-elimination algorithm w.r.t. the set X).

Proof. First, we recall from Proposition 29 that H is a bounded least general generalization of all clauses from \mathcal{A} w.r.t. X . Now, we assume (for contradiction) that there is a clause $H^* \in X$ which θ -subsumes all clauses from \mathcal{A} and no clauses from \mathcal{B} and that, at the same time, there is either a clause $A \in \mathcal{A}$ not θ -subsumed by H or a clause $B \in \mathcal{B}$ θ -subsumed by H . Neither of these possibilities can happen. Since H is a bounded least general generalization of all clauses from \mathcal{A} , it must θ -subsume all of these clauses, so the first possibility cannot happen. The other possibility can be ruled out as follows. If there is a clause $B \in \mathcal{B}$ such that $H \preceq_\theta B$ then also $H^* \preceq_\theta B$ because $H^* \in X$ and for any $C \in X$ which θ -subsumes all clauses from \mathcal{A} it must hold $C \preceq_\theta H$ (from definition of bounded least general generalization), therefore $H^* \preceq_\theta H$ would also have to be true, which would imply $H^* \preceq_\theta B$. This is again a contradiction with H^* not subsuming any clause from \mathcal{B} . \square

Proposition 30. *Let X be a set of clauses and let \mathcal{E}^+ and \mathcal{E}^- be sets of positive and negative examples, respectively. Next, let the procedure $\text{CandidateExamples}(H, \mathcal{E}^+)$ (used by the algorithm S-BULL) always return the set \mathcal{E}^+ . If there is a clause $H \in X$ which covers $p > 1$ positive examples and n negative examples then the algorithm S-BULL always finds a clause which covers p' positive examples and n' negative examples and it holds $p' - n' \geq p - n$.*

Proof. Let \mathcal{E}_H^+ be the set of positive examples covered by H . Since $H \in X$, we can infer using Proposition 29 that any clause obtained as a bounded least general generalization w.r.t. the set X of the clauses in \mathcal{E}_H^+ must necessarily θ -subsume exactly the positive examples covered by H (i.e. all $E \in \mathcal{E}_H^+$), no other positive examples and a subset of negative examples covered by H . Now, we show that bounded least general generalization w.r.t. X of the clauses contained in the set \mathcal{E}_H^+ must be computed and added to the list Open by S-BULL at some point. This is not as obvious as one might think because of the inner-most loop of the algorithm S-BULL in which bounded least general generalization of the examples covered by the current hypothesis w.r.t. the relation \triangleleft_X is computed. This loop is, however, essential for guaranteeing that the positive examples covered w.r.t. the x -presubsumption relation \triangleleft_X will also be covered w.r.t. the ordinary θ -subsumption. We will prove that the bounded least general generalization of the set \mathcal{E}_H^+ will be computed at some point by the S-BULL algorithm. Let us assume, for contradiction, that the bounded least general generalization of the set \mathcal{E}_H^+ is never actually computed and let $\mathcal{A} \subseteq \mathcal{E}_H^+$ be the maximal set of clauses for which LGG_X was computed and added to the list Open . Let us now take a clause $E \in (\mathcal{E}_H^+ \setminus \mathcal{A})$. It must be the case that S-BULL computes LGG_X of clauses in \mathcal{A} and the clause E which we denote as $H_{\mathcal{A} \cup \{E\}}$ (this is obvious from quick inspection of the pseudocode in Algorithm 6). Since \mathcal{A} is the maximal subset of \mathcal{E}_H^+ for which LGG_X was computed and added to the list Open , it must also be the case that $H_{\mathcal{A} \cup \{E\}}$ is replaced (in the inner-most loop of S-BULL) by LGG_X of all positive examples covered w.r.t. the relation \triangleleft_X by $H_{\mathcal{A} \cup \{E\}}$ and this set must contain at least one clause not contained in \mathcal{E}_H^+ (from maximality of \mathcal{A}). However, this is not possible for the following reason. If LGG_X of the set $\mathcal{A} \cup \{E\}$ covered,

w.r.t. the relation \triangleleft_X , a set \mathcal{B} which is not a subset of \mathcal{E}_H^+ then it would also x -subsume the examples in this set. This would mean that any clause from the set X which would θ -subsume all clauses from $\mathcal{A} \cup \{E\}$ would also have to θ -subsume all clauses from the set \mathcal{B} , therefore the clause H would also have to θ -subsume all clauses from \mathcal{B} but this would be a contradiction with the assumption that the clause H θ -subsumes only the clauses from the set \mathcal{E}_H^+ . So far we have managed to show that the bounded least general generalization of the positive examples covered by H must be computed at some point by the S-BULL algorithm. It remains to be shown that either this bounded least general generalization or a clause with score at least as high as the clause H will be returned by S-BULL. In other words, what we need to show is that it does not matter that S-BULL computes its score, which is used to compare candidate clauses, using the x -presubsumption relation \triangleleft_X .

First, notice that when `Score` is computed on line 25 of Algorithm 6, the set `PosCovered` is equal to the set of positive examples θ -subsumed by the current candidate hypothesis H^* . This is ensured by the fact that H^* is an LGG_X of all the positive examples it covers w.r.t. the x -presubsumption relation (due to the inner-most loop on line 16 of the algorithm). Furthermore, the set `NegCovered` is a superset of the θ -subsumed negative examples (because x -presubsumption is always implied by θ -subsumption). The number `Score` computed by the algorithm using x -presubsumption \triangleleft_X of the examples in \mathcal{E}_H^+ is therefore bounded from above by the “true” score that would be computed using θ -subsumption. Moreover, the score of a clause H' constructed as LGG_X of the positive examples θ -subsumed by H is bounded from below by $p - n$ because H' covers the same set of positive examples as H and a subset of the negative examples θ -subsumed by H as was already discussed above. So when the clause H' is compared with another clause H'' and the other clause H'' has higher score computed using the x -presubsumption relation then this other clause H'' must be also better than H according to the “true” score computed using θ -subsumption (because its true score is higher than its score computed by x -presubsumption). More formally, if we denote by $\text{score}_{\triangleleft_X}(X)$ the score of a clause X computed using the x -presubsumption relation \triangleleft_X and by $\text{score}_{\preceq_\theta}(X)$ the score of a clause X computed using θ -subsumption then we have:

$$\text{score}_{\preceq_\theta}(H'') \geq \text{score}_{\triangleleft_X}(H'') \geq \text{score}_{\triangleleft_X}(H') \geq \text{score}_{\triangleleft_X}(H) = \text{score}_{\preceq_\theta}(H).$$

This finishes the proof because we have already shown that a LGG_X of the positive examples covered by H must be computed at some point and that if it is not selected as the best candidate clause (`BestClause`) then the selected clause must be actually at least as good. \square

The next lemma giving a sufficient condition for a relation to be an x -presubsumption will be useful for proving that certain procedures are x -presubsumptions.

Lemma 9. *Let X be a set of clauses and \triangleleft_X be a relation satisfying the following conditions:*

1. *If $A \triangleleft_X B$ and $C \subseteq A$ then $C \triangleleft_X B$.*
2. *If $A \in X$, ϑ is a substitution and $A\vartheta \triangleleft_X B$ then $A \preceq_\theta B$.*
3. *If $A \preceq_\theta B$ then $A \triangleleft_X B$.*

Then \triangleleft_X is an x -presubsumption w.r.t. the set X .

Proof. We show that any relation \triangleleft_X satisfying the conditions of the proposition also satisfies the conditions on x -presubsumption stated in Definition 35.

1. *If $A \in X$ and $A \triangleleft_X B$ then $A \preceq_\theta B$:* This follows from the condition 2 where we set ϑ to be an identity substitution.
2. *If $A \preceq_\theta B$ then $A \triangleleft_X B$:* This follows (in fact, it is equivalent) to the condition 3.

3. If $A \in X$, $A \triangleleft_X B$ and $B \triangleleft_X C$ then $A \triangleleft_X C$: We assume that $A \in X$, $A \triangleleft_X B$ and $B \triangleleft_X C$ because the implication is otherwise vacuously true. Then $A \preceq_\theta B$ (using the condition 2 because $A \in X$) therefore there is a substitution ϑ such that $A\vartheta \subseteq B$. We can infer $A\vartheta \triangleleft_X C$ from it using the condition 1. Finally, we get $A \preceq_\theta C$ from it using the condition 2.

□

Proposition 31. Let $k \in \mathbb{N}$ and let \triangleleft_k be a relation on clauses defined as follows: $A \triangleleft_k B$ if and only if the k -consistency algorithm run on the CSP-encoding (described in Section 7.1) of the θ -subsumption problem $A \preceq_\theta B$ returns true. The relation \triangleleft_k is an χ -presubsumption w.r.t. the set X_k of all clauses with treewidth at most k .

Proof. We need to verify that \triangleleft_k satisfies the conditions stated in Lemma 9

1. If $A \triangleleft_k B$ and $C \subseteq A$ then $C \triangleleft_k B$. This holds because if the k -consistency algorithm returns true for a problem then it must also return true for any of its subproblems (recall the discussion in Section 7.1). It is easy to check that if $C \subseteq A$ are clauses then the CSP problem encoding the θ -subsumption problem $C \preceq_\theta B$ is a subproblem of the CSP encoding of the θ -subsumption problem $A \preceq_\theta B$. Therefore this condition holds.
2. If $A \in X$, ϑ is a substitution and $A\vartheta \triangleleft_k B$ then $A \preceq_\theta B$. The CSP encoding of the problem $A \preceq_\theta B$ is a subproblem of the problem encoding $A\vartheta \preceq_\theta B$, in which there are additional constraints enforcing consistency with the substitution ϑ (because the set of constraints of the former is a subset of the constraints of the latter). Therefore if $A\vartheta \triangleleft_k B$ then also $A \triangleleft_k B$ and, since $A \in X$, it also holds $A \preceq_\theta B$.
3. If $A \preceq_\theta B$ then $A \triangleleft_k B$. This is a property of k -consistency (recall the discussion in Section 7.1).

□

The next proof is only a slight variant of the proof of the previous proposition since the properties of k -consistency and generalized arc-consistency are analogical to each other. What differs is mostly the set X w.r.t. which k -consistency and generalized arc-consistency can be used as χ -presubsumptions.

Proposition 32. Let \triangleleft_{GAC} be a relation on clauses defined as follows: $A \triangleleft_{GAC} B$ if and only if the generalized arc-consistency algorithm run on the CSP-encoding (described in Section 7.1) of the θ -subsumption problem $A \preceq_\theta B$ returns true. The relation \triangleleft_{GAC} is an χ -presubsumption w.r.t. the set X_A of all acyclic clauses.

Proof. We need to verify that \triangleleft_{GAC} satisfies the conditions stated in Lemma 9.

1. If $A \triangleleft_{GAC} B$ and $C \subseteq A$ then $C \triangleleft_k B$. This holds because if the generalized arc-consistency algorithm returns true for a problem then it must also return true for any of its subproblems (recall the discussion in Section 7.1). It is easy to check that if $C \subseteq A$ are clauses then the CSP problem encoding the θ -subsumption problem $C \preceq_\theta B$ is a subproblem of the CSP encoding of the θ -subsumption problem $A \preceq_\theta B$. Therefore this condition holds.
2. If $A \in X$, ϑ is a substitution and $A\vartheta \triangleleft_{GAC} B$ then $A \preceq_\theta B$. The CSP encoding of the problem $A \preceq_\theta B$ is a subproblem of the problem encoding $A\vartheta \preceq_\theta B$, in which there are additional constraints enforcing consistency with the substitution ϑ (because the set of constraints of the former is a subset of the constraints of the latter). Therefore if $A\vartheta \triangleleft_{GAC} B$ then also $A \triangleleft_{GAC} B$ and, since $A \in X$, it also holds $A \preceq_\theta B$.
3. If $A \preceq_\theta B$ then $A \triangleleft_{GAC} B$. This is a property of generalized arc-consistency (recall the discussion in Section 7.1).

□

In the proof of Proposition 33, we will need the following simple lemma stating the intuitively obvious fact that every treelike hypergraph with hyperedges of arity 2 is a forest.

Lemma 10. *If a treelike hypergraph G has only hyperedges of arity 2 and no loops then it is a forest.*

Proof. Hypergraphs with hyperedges of arity 2 are graphs, so we can use graph-theoretic concepts in this proof. First, if G is treelike then there must be a sequence $\mathcal{S} = S_1, S_2, \dots, S_k$ of the reduction steps from Definition 38 which reduces G to the empty graph. We will now use induction on length of the sequence \mathcal{S} to show that G must be a forest. The base case is obvious. If $|\mathcal{S}| = 1$ then G must consist of a single vertex (such a graph is a forest). We will now assume the induction hypothesis that every graph which can be reduced to the empty graph using a sequence of at most n reduction steps is a forest and we will show that then every graph which can be reduced to the empty graph by a sequence of $n + 1$ steps must be a forest as well. Let G be reducible to the empty graph using a sequence of $n + 1$ steps. Let us apply the first rule S_1 of this sequence. The resulting graph may be slightly deficient as it may have an edge with only one vertex incident to it (which is not an edge in the graph-theoretic sense but which is an edge in the hypergraph-theoretic sense) – we can immediately remove such an edge by the application of the reduction rule which removes an edge which contains fewer than two vertices. In any case, as a result, we get a graph which can be reduced to the empty graph in at most n steps for which we already know that it is a forest. It remains to be shown that the original graph then must have been a forest as well. We may notice that S_1 must be an application of the rule which removes a vertex which is contained in at most one edge (this edge is not a loop) because the hypergraphs, either the input hypergraph or the intermediate hypergraphs appearing in the inductive steps, are graphs which is also ensured by the fact that we always remove the edges containing just one vertex – therefore S_1 cannot be an application of the rule which removes an edge with at most one vertex. The rule which removes a vertex can be used only if the vertex is contained in at most one edge, therefore we know that there is at most one edge which connects that vertex to the rest of the graph about which we know that it is a forest. Therefore also this bigger graph must be a forest. This establishes the inductive step.

□

The proof of the next proposition might be considered to be too long given that it establishes a property which is intuitively clear. The most space is spent by proving that the alternative CSP representation introduced in Section 7.6.2 is treelike for treelike clauses. This proof might actually be shorter if we decided to define treelike clauses in a different way, however, that might make the presentation in the main text less coherent. We preferred the coherence of the main text over the length of this particular proof.

Proposition 33. *Let \triangleleft_1 be a relation on clauses defined as follows: $A \triangleleft_1 B$ if and only if the k -consistency algorithm with $k = 1$, i.e. the 1-consistency algorithm, run on the alternative CSP-encoding (described in Section 7.6.2) of the θ -subsumption problem $A \preceq_\theta B$ returns true. The relation \triangleleft_1 is an χ -presubsumption w.r.t. the set X_\top of all treelike clauses.*

Proof. It suffices to show that if a clause A is treelike then the alternative CSP encoding of any problem $A \preceq_\theta B$ has treewidth 1 and then use the same reasoning that was used in the proof of Proposition 31 limited to $k = 1$.

Let us have a θ -subsumption problem $A \preceq_\theta B$, let \mathcal{P} be its alternative CSP encoding and let G be the Gaifman graph of \mathcal{P} (note that Gaifman graphs never contain loops). Let $\mathcal{S} = S_1, S_2, \dots, S_n$ be a sequence of steps of the iteration procedure from Definition 38 which produces the empty clause from A . We will show how to find another sequence \mathcal{S}' of steps which produces the empty graph from G (which will show that it is treelike). We go through the sequence of steps $S_i \in \mathcal{S}$ and perform the following steps. If S_i is a step which removes a literal l with arity smaller or equal to 1 from A then we find the respective vertex corresponding to the CSP variable V_l and

remove it and all its incident edges from G . If S_i is a step which removes a variable V from A which is contained in at most one literal then we remove the vertex corresponding to the CSP variable V_V from G and all the edges incident to it. We need to show two things now: (i) that the result of the procedure is the empty graph and (ii) that the operations performed in the given order can be used at the given times according to rules from Definition 38, i.e. that if a vertex and all its incident edges are removed from the graph G then that vertex has actually only one incident edge (which is then also removed).

We start by proving (i), i.e. that the result of the application of the rules in the sequence S' on the graph G produces the empty graph. This is obvious since the vertices in G can be divided into two disjoint sets such that there is a one-to-one correspondence between the vertices in the first set and the variables in $\text{vars}(A)$ and between the vertices in the second set and the literals in A . Since all the literals and variables of A are removed in the end and since, always, when a literal or a variable is removed from A , a vertex corresponding to that variable or literal is removed from G , it follows that G must be empty in the end as well.

Now, we will prove (ii), i.e. that the steps from sequence S' can be applied on the graph G . We will use induction on the length of the sequence S . We start with the base case $|S| = 1$. In this case there must be just one literal in A and it must have arity 0 (there cannot be a variable without a literal). So, the corresponding graph G must contain just one vertex and no edge and this vertex can be therefore removed in accordance with the rules from Definition 38. Now, we prove the induction step assuming the induction hypothesis that all rules in any sequence S' of length n , which produces the empty graph from G , can be applied in the given order while satisfying the conditions from Definition 38. Let us have a clause A which is treelike and which can be converted to the empty clause by $n + 1$ iterative applications of rules from a sequence S_{n+1} , complying with Definition 38. We take the first step S_1 from the sequence S_{n+1} and apply it on A (which gives us a clause which can be reduced to the empty clause in n steps). If S_1 removes a variable $V \in \text{vars}(A)$ contained in at most one literal then the operation that we perform on G is that we find the respective vertex in G corresponding to the CSP-variable V_V and remove it and all its incident edges from G . The question is whether we can do that. We can because the removed vertex must be incident to only one edge (which connects it to the vertex representing the CSP-variable corresponding to the literal in which the variable V was contained). The incident edge can also be removed because, after the removal of the vertex, it contains only one vertex (and therefore can be removed). Similarly, if S_1 removes a literal l which contains fewer than 2 variables, the operation applied on G is the removal of the vertex corresponding to the CSP-variable V_l and of all its incident edges. This can be done because the vertex corresponding to the CSP-variable V_l can be connected only to one other vertex by an edge because the literal l has arity at most one and therefore there is at most one constraint in \mathcal{P} involving V_l which connects it to the CSP-variable representing the only variable in the arguments of l . It follows that we can always apply the first rule in S' on G which gives us a smaller problem with hypergraph that can be converted to the empty hypergraph by a sequence of length n by the induction hypothesis. Therefore if a clause A is treelike then the Gaifman graph of the alternative CSP encoding of any θ -subsumption problem of the type $A \preceq_{\theta} B$ must also be treelike.

Once we know that the graph G is treelike (when treated as a hypergraph with at most binary hyperedges), we can invoke Lemma 10 which states that every treelike hypergraph with at most binary hyperedges is a tree or a forest. Since trees and forests have treewidth 1, we can follow the exact reasoning in the proof of Proposition 31 to show that 1-consistency is an x -presubsumption w.r.t. the set of all treelike clauses. \square

Proposition 34. *Let \mathcal{LB} be a language bias and let $X_{\mathcal{LB}}$ be the set of all clauses complying with \mathcal{LB} . Let A be a clause. If $A_{\mathcal{LB}} \subseteq A$ is a clause composed of exactly the literals from A complying with \mathcal{LB} then $A_{\mathcal{LB}} \preceq_{\theta} A$ and $A \not\preceq_X A_{\mathcal{LB}}$ w.r.t. the set $X_{\mathcal{LB}}$.*

Proof. The first part of the proposition is obvious. If $A_{\mathcal{LB}} \subseteq A$ then $A_{\mathcal{LB}} \preceq_{\theta} A$. We show the validity of the second part by contradiction. We assume that $A \preceq_X A_{\mathcal{LB}}$. This means that there

is a clause $C \in X_{\mathcal{L}\mathcal{B}}$ such that $C \preceq_{\theta} A$ and $C \not\preceq_{\theta} A_{\mathcal{L}\mathcal{B}}$. Let ϑ be a substitution such that $C\vartheta \subseteq A$. The substitution ϑ can map only literals complying with $\mathcal{L}\mathcal{B}$ to literals also complying with $\mathcal{L}\mathcal{B}$ (because constants cannot be mapped to variables) so $C\vartheta \subseteq A_{\mathcal{L}\mathcal{B}}$ which also means $C \preceq_{\theta} A_{\mathcal{L}\mathcal{B}}$. This is a contradiction with $C \not\preceq_{\theta} A_{\mathcal{L}\mathcal{B}}$. \square

Proposition 35. *Let $\mathcal{L}\mathcal{B}$ be a language bias and let $X_{\mathcal{L}\mathcal{B}}$ be the set of all clauses complying with $\mathcal{L}\mathcal{B}$. Let A_1, A_2, \dots, A_n be clauses and let $B = \text{LGG}(A_1, A_2, \dots, A_n)$. If $B_{\mathcal{L}\mathcal{B}}$ is a clause obtained by removing from B all literals which do not comply with $\mathcal{L}\mathcal{B}$ then $B_{\mathcal{L}\mathcal{B}}$ is a bounded LGG w.r.t. the set $X_{\mathcal{L}\mathcal{B}}$. If, in addition, X is a set of clauses then $\text{litelim}_X(B_{\mathcal{L}\mathcal{B}})$ is a bounded LGG w.r.t. the set $X \cap X_{\mathcal{L}\mathcal{B}}$.*

Proof. Follows easily from Proposition 27 and Proposition 34. \square

7.11 COMPLEXITY OF BOUNDED SUBSUMPTIONS AND REDUCTIONS

In this section, we briefly discuss complexity of computing x -subsumptions and x -reductions. The main result presented in this section is that finding a maximally reduced clause w.r.t. a set X may be NP-hard even if a polynomial-time procedure for deciding x -presubsumption is available w.r.t. X .

We start with the questions regarding complexity of x -subsumptions and x -reductions w.r.t. finite sets. What determines complexity in this case is whether the set X is part of the input or is fixed.

Proposition 36. *Let X be a fixed finite set. The problem $A \preceq_X B$ (i.e. deciding x -subsumption w.r.t. X) can be solved in time polynomial in $|A|$ and $|B|$.*

Proof. The x -subsumption problem can be decided by checking x -subsumption straightforwardly according to the definition (Definition 34). First, we find the subset of all clauses $C \in X$ which θ -subsume A (i.e. clauses $C \in X$ such that $C \preceq_{\theta} A$). This can be achieved using some exponential-time θ -subsumption algorithm in time $\mathcal{O}(|X| \cdot \max_{C \in X} |A|^{|C|})$, which is polynomial in $|A|$. Similarly, we can check whether all these clauses also θ -subsume the clause B in time which is polynomial in $|B|$. If they all θ -subsume B then we output *true*, otherwise we output *false*. So, the algorithm runs in time polynomial in $|A|$ and $|B|$ and decides x -subsumption correctly. \square

Thus, deciding x -subsumption $A \preceq_X B$ w.r.t. a fixed finite set can be solved in time polynomial in the sizes of the clauses A and B . However, the algorithm is far from being practical as the hidden multiplicative constant may be quite large as it depends exponentially on the size of the largest clause in the set X .

If the set X is not fixed but is finite and part of the input then the problem of deciding x -subsumption w.r.t. X becomes NP-hard.

Proposition 37. *The problem of deciding x -subsumption $A \preceq_X B$ w.r.t. a finite set X , where A , B and X are part of the input, is NP-hard.*

Proof. We show this by a simple reduction from the NP-hard θ -subsumption problem. If we have a θ -subsumption problem $A \preceq_{\theta} B$ then we can convert it to an x -subsumption problem with finite X as follows. We set $X = \{A\}$ and construct an x -subsumption problem $A \preceq_X B$. From definition of x -subsumption it must hold: $(A \preceq_X B) \Leftrightarrow (\forall C \in X : (C \preceq_{\theta} A) \Rightarrow (C \preceq_{\theta} B))$ which is equivalent to $(A \preceq_X B) \Leftrightarrow ((A \preceq_{\theta} A) \Rightarrow (A \preceq_{\theta} B))$ (because $C = \{A\}$) which can be further simplified as $(A \preceq_X B) \Leftrightarrow (A \preceq_{\theta} B)$. Thus, we are able to convert the NP-hard θ -subsumption problem to the x -subsumption problem with a finite set X which must be therefore NP-hard as well. \square

We now turn our attention to x -reductions w.r.t. finite sets of clauses where, as we shall see, the situation is analogous to the case of x -subsumption w.r.t. finite sets. When the set X is finite and fixed, x -reduction can be computed in time which is polynomial in the size of the clause to be reduced. If X is finite but not fixed, i.e. part of input, then the problem becomes NP-hard.

Proposition 38. *Let X be a fixed finite set. An x -reduction of a clause A w.r.t. the set X can be computed in time polynomial in $|A|$.*

Proof. Let us assume, without loss of generality, that the clauses in X are standardized apart (i.e. that for any two clauses $C_1, C_2 \in X$, $C_1 \neq C_2$ it holds $\text{vars}(C_1) \cap \text{vars}(C_2) = \emptyset$). Let X^+ be the set of all $C_i \in X$ s.t. $C_i \preceq_\theta A$. This set can be constructed in time which is polynomial in the size of A (though, exponential in the sizes of the clauses in C_i which are nevertheless fixed and there is only a finite number of them). Let ϑ be a substitution such that $C_i\vartheta \subseteq A$ for all $C_i \in X^+$ and $|\bigcup_{C_i \in X^+} C_i\vartheta|$ is minimal. The substitution ϑ can be found in time polynomial in the size of A by enumeration of possible substitutions which map literals appearing in clauses contained in X^+ to literals in A because the number of such substitutions is bounded by $|A|^{\sum_{C_i \in X^+} |C_i|}$ (which is polynomial in $|A|$). Moreover $\bigcup_{C_i \in X^+} C_i\vartheta$ is an x -reduction of A which can be seen from the fact that $\bigcup_{C_i \in X^+} C_i\vartheta \subseteq A$ and $A \preceq_X \bigcup_{C_i \in X^+} C_i\vartheta$ because every clause $C_i \in X$ which θ -subsumes A must also, by construction, θ -subsume the clause $\bigcup_{C_i \in X^+} C_i\vartheta$. \square

Again, despite being polynomial, the algorithm outlined in the proof of the above proposition is hardly practical. It serves mainly to point out that the problem is not NP-hard when X is finite and fixed rather than showing that it is *practically tractable*.

Likewise in the case of x -subsumption, when the set X is not fixed but is part of the input, the problem of finding an x -reduction becomes NP-hard.

Proposition 39. *The problem of deciding whether a clause A is x -reducible w.r.t. a finite set X , where A and X are part of the input, is NP-hard.*

Proof. We show this by finding a reduction from the NP-hard θ -reducibility problem. Let us have a clause A for which we would like to check whether it is θ -reducible. We now construct an x -reducibility problem equivalent to the θ -reducibility problem. We set $X = \{A\}$ and check whether A is x -reducible w.r.t. the set X . What we need to show is that A will be x -reducible w.r.t. the set X if and only if A is θ -reducible. If A is θ -reducible then it is also x -reducible w.r.t. X because θ -subsumption always implies x -subsumption. It remains to show also the opposite. If A is x -reducible, there must be a smaller clause $A' \subseteq A$ such that $A \preceq_X A'$. Therefore, from definition of x -subsumption, it must also hold $\forall C \in X : (C \preceq_\theta A) \Rightarrow (C \preceq_\theta A')$ which is equivalent to $(A \preceq_\theta A) \Rightarrow (A \preceq_\theta A')$ (because $X = \{A\}$) and therefore $A \preceq_\theta A'$. Since A' is a subset of A and A' is smaller than A , A must be θ -reducible. We have therefore shown also the other direction of the implication. It follows that checking x -reducibility of a clause is NP-hard when the set X is part of the input. \square

Since the decision version of the problem of finding x -reductions, i.e. checking x -reducibility of clauses, is NP-hard when the set X is part of the input, so also the optimization version of the problem must be NP-hard, i.e. finding an x -reduction (rather than just checking x -reducibility).

Now, we turn our attention to the case when the set X is infinite. When X is infinite, we often have only an efficiently computable x -presubsumption, but not the x -subsumption. Therefore, for practical reasons, we are more concerned with the question how hard it is to find a *maximally reduced clause* using x -presubsumptions, rather than with the analogous question about hardness of finding full x -reductions. We show that finding the *maximally reduced clause w.r.t. x -presubsumption* may be NP-hard even if the x -presubsumption can be checked in polynomial time. We do this in several steps by finding a reduction from the problem known as *minimum equivalent digraph*, which is NP-hard [5].

First, we need to define the problem of finding a *maximally reduced clause w.r.t. an x -presubsumption* and the problem of finding a *minimum equivalent digraph*.

Definition 40. *Let A be a clause and \triangleleft_X be an x -presubsumption w.r.t. some set X . A clause \hat{A} is said to be maximally reduced w.r.t. the x -presubsumption \triangleleft_X if and only if $\hat{A} \preceq_\theta A$, $A \triangleleft_X \hat{A}$ and $|\hat{A}|$ is minimal.*

Definition 41 (Minimum equivalent digraph problem). *The minimum equivalent digraph problem is the following optimization problem: Find a maximum-cardinality subset of edges $E' \subseteq E$ of a digraph $G = (V, E)$ such that there is a directed path from vertex v to vertex w in the new graph $G' = (V, E')$ if and only if there is a directed path from v to w in the original graph G .*

We distinguish *paths*, which contain no repeated edges, and *walks*, which may contain repeated edges.

Next, we define the set X^* which will consist of so-called *walk-clauses* and will be used for showing that computing maximally reduced clauses w.r.t. an χ -presubsumption may be NP-hard even if the χ -presubsumption itself is checkable in polynomial time. *Walk-clauses* are clauses of the form

$$\begin{aligned} C_1^{vw} &= \text{edge}(v, w) \vee \text{edge}(v, v) \vee \text{edge}(w, w) \vee s(Y_1) \vee c(Y_1, Y_2) \vee e(Y_2) \\ C_2^{vw} &= \text{edge}(v, A_1) \vee \text{edge}(v, v) \vee \text{edge}(A_1, A_1) \vee \text{edge}(A_1, w) \vee \text{edge}(w, w) \vee \\ &\quad \vee s(Y_1) \vee c(Y_1, Y_2) \vee c(Y_2, Y_3) \vee e(Y_3) \\ &\dots \\ C_n^{vw} &= \text{edge}(v, A_1) \vee \text{edge}(v, v) \vee \text{edge}(A_1, A_1) \vee \text{edge}(A_1, A_2) \vee \text{edge}(A_2, A_2) \vee \\ &\quad \vee \text{edge}(A_{n-1}, w) \vee \text{edge}(w, w) \vee \dots \vee s(Y_1) \vee c(Y_1, Y_2) \vee \dots \vee c(Y_n, Y_{n+1}) \vee \\ &\quad \vee e(Y_{n+1}) \end{aligned}$$

The literals $\text{edge}(A, B)$ will be used for representing edges in a directed walk. The literals $s(\dots)$, $c(\dots)$ and $e(\dots)$ will be used to distinguish walks of different lengths from each other. The set X^* is the set of all such walk-clauses.

Now, we describe an χ -presubsumption \triangleleft_* w.r.t. X^* suitable for showing NP-hardness of the problem of finding a maximally reduced clause w.r.t. to \triangleleft_* .

Algorithm \triangleleft_* :

1. Given clauses A and B .
2. Create two copies A' and A'' of A , two copies B' and B'' of B and standardize them apart.
3. Remove all literals other than those based on predicate $\text{edge}/2$ from A' and B' and replace A' and B' by the respective newly created clauses.
4. Remove all literals other than those based on predicates $c/2$, $s/1$ and $e/1$ from A'' and B'' and replace A'' and B'' by the respective newly created clauses.
5. Use the χ -presubsumption \triangleleft_1 from Section 7.6.1 (i.e. the χ -presubsumption based on 1-consistency) to check $A'' \triangleleft_1 B''$. If the result is *false*, return *false*, otherwise continue to the next step.
6. Construct a graph $G_A = (V_A, E_A)$ from A' and a graph $G_B = (V_B, E_B)$ from B' as follows. Introduce one vertex for every term t in A (B , respectively). If t is a constant, label the corresponding vertex by the constant. Add a directed edge $e = (v_{t_1}, v_{t_2})$ for every two vertices v_{t_1}, v_{t_2} corresponding to terms t_1, t_2 for which there is a literal $e(t_1, t_2)$ in A (B , respectively).
7. Remove from G_A and G_B all vertices which do not have loops incident to them (a loop is a directed edge with the same start-vertex and end-vertex).
8. For all $i \in \{|A''| - 2, |A''| - 1, \dots, \max\{|V_A|, |V_B|\} + 1\}$, find a set L_A^i of all pairs of labels of vertices of the graph G_A which are connected by a directed walk of length i .
9. Similarly, for all $i \in \{|A''| - 2, |A''| - 1, \dots, \max\{|V_A|, |V_B|\} + 1\}$, find a set L_B^i of all pairs of labels of vertices of the graph G_B which are connected by a directed walk of length i .

10. If $L_A^i \subseteq L_B^i$ for all $i \in \{|A''| - 2, |A''| - 1, \dots, \max\{|V_A|, |V_B|\} + 1\}$, return *true*, otherwise return *false*.

In what follows, if A or B or C are clauses then A' , B' , C' or A'' , B'' or C'' will denote the respective clauses constructed in Algorithm \triangleleft_* .

What we need to show now is that the above algorithm, indeed, represents an x -presubsumption w.r.t. the set X^* and that it runs in polynomial time. This is shown formally in Lemma 13.

Lemma 11. *If $A = C_n^{vw} \in X^*$, B is a clause, and the graph G_B associated to the clause B contains an oriented walk of length n from v to w then $A' \preceq_\theta B'$.*

Proof. If G_B contains an oriented walk of length n from v to w then there must be a set of literals in B' of at least one of the following types:

1. $\text{edge}(v, w) \vee \text{edge}(v, v) \vee \text{edge}(w, w)$
2. $\text{edge}(v, t_1) \vee \text{edge}(t_1, t_1) \vee \text{edge}(t_1, t_2) \vee \dots \vee \text{edge}(t_m, w) \vee \text{edge}(w, w)$

Here, $m \leq n$ and any t_i may be either a variable or a constant. It follows rather easily that $A' \preceq_\theta B'$. \square

Lemma 12. *Let A be a clause and let G_A be the respective graph constructed by the algorithm for checking the x -presubsumption \triangleleft_* . If G_A contains an oriented walk of length n from v to w then for any $m > n$ there is a walk from v to w of length m .*

Proof. This follows from the fact that every vertex in G_A has a loop incident to it. \square

Lemma 13. *The relation given by pairs of clauses, for which Algorithm \triangleleft_* returns *true*, is an x -presubsumption w.r.t. the set X^* . The algorithm runs in time polynomial in the size of the clauses A and B .*

Proof. We need to verify that Algorithm \triangleleft_* satisfies the conditions from Definition 35.

1. *If $A \in X^*$ and $A \triangleleft_* B$ then $A \preceq_\theta B$:* Let us assume that $A = C_n^{vw} \in X^*$ and $A \triangleleft_* B$ because the implication is otherwise vacuously true. It must hold $A'' \preceq_\theta B$ because $A'' \triangleleft_1 B$ and \triangleleft_1 is an x -presubsumption w.r.t. the set of clauses with treewidth 1 and A'' has treewidth 1. It remains to show that also $A' \preceq_\theta B$ (recall that $\text{vars}(A') \cap \text{vars}(A'') = \emptyset$). Obviously, there must be a walk in G_A from v to w of length n . There must be also a walk from v to w of the same length in G_B , because $A \triangleleft_* B$, and therefore $A' \preceq_\theta B'$ (using Lemma 11) and consequently also $A \preceq_\theta B$ (recalling that $A'' \preceq_\theta B$).
2. *If $A \preceq_\theta B$ then $A \triangleleft_* B$:* If $A \preceq_\theta B$ then the value *false* cannot be returned on line 5 of the algorithm because the x -presubsumption \triangleleft_1 , which is used there, also satisfies $(A \preceq_\theta B) \Rightarrow (A \triangleleft_1 B)$. So we need to rule out only the possibility that *false* will be returned on line 10 of the algorithm. This can be shown as follows. If $A \preceq_\theta B$ then there must be a substitution θ such that $A\theta \subseteq B$. Therefore the graph G_A must be homomorphic to the graph G_B (even if we consider the removal of loop-free vertices, which is actually easy to check). It follows that any walk in G_A can be mapped on a walk of the same length in G_B (the number of repeated edges may differ). Therefore, it must also hold $L_A^i \subseteq L_B^i$ for all relevant i 's (i.e. for every $i \in \{|A''| - 2, |A''| - 1, \dots, \max\{|V_A|, |V_B|\} + 1\}$) (recall Lemma 12) and the value returned by the algorithm must be *true*.
3. *If $A \in X^*$, $A \triangleleft_* B$ and $B \triangleleft_* C$ then $A \triangleleft_* C$:* We assume that $A = C_n^{vw} \in X^*$, $A \triangleleft_* B$ and $B \triangleleft_* C$ and show that then, necessarily, the algorithm \triangleleft_* checking $A \triangleleft_* C$ must return *true*. It certainly will not return *false* on line 5 because \triangleleft_1 satisfies this condition w.r.t. the set of all clauses with treewidth 1 and because A'' has treewidth 1. So, we need only to show that *true* will be returned on line 10, i.e. to show that $L_A^i \subseteq L_C^i$ for every $i \in \{1, 2, \dots, \max\{|V_A|, |V_C|\} + 1\}$. Furthermore, $L_A^i = \emptyset$ for all $i < n$ and $L_A^i = \{(v, w)\}$ for all

$i \geq n$. Since $A \triangleleft_* B$, it must hold $(v, w) \in L_B^i$ for all $i \geq n$ (using Lemma 12 for showing this for $i > \max\{|V_A|, |V_B|\} + 1$). Similarly, we can use the fact $B \triangleleft_* C$ to show $(v, w) \in L_C^i$ for all $i \geq n$. Thus, the value *true* must be returned on line 10 because $L_A^i \subseteq L_C^i$ for all $i \in \{|A''| - 2, |A''| - 1, \dots, \max\{|V_A|, |V_C|\} + 1\}$.

We also need to show that the algorithm \triangleleft_* runs in polynomial time but that is easy. The only part which might need a more detailed discussion is the computation of all pairs of vertices connected by an oriented walk. Computing this can be done in polynomial time by breadth-first search. \square

Now, we need to show that the minimum equivalent digraph problem can be reduced to finding a maximally reduced clause w.r.t. the x -presubsumption \triangleleft_* . The reduction procedure accomplishing this is given next.

Algorithm ClGrRed:

1. Given a directed graph $G = (V, E)$.
2. Construct a clause C' such that:
 - a) There is a literal $\text{edge}(v, w)$, where v and w are constants, for every $(v, w) \in E$.
 - b) There is a literal $\text{edge}(v, v)$, where v is a constant, for every vertex $v \in V$.
3. Construct a clause $C'' = s(Y_1) \vee c(Y_1, Y_2) \vee \dots \vee c(Y_n, Y_{n+1}), e(Y_{n+1})$ where $n = |V|$.
4. Set $C := C' \cup C''$.
5. Find a maximally reduced clause C_{red} for C w.r.t. the x -presubsumption \triangleleft_* .
6. Create a graph $G' = (V, E')$ where for any $v, w \in V$, $(v, w) \in E'$ if and only $(v, w) \in E$ and C_{red} contains a literal $\text{edge}(v, w)$.
7. Return G' .

Lemma 14. *Let G be a directed graph and let G' be a graph computed from G by Algorithm ClGrRed. Then G' is a minimum directed graph equivalent to G .*

Proof. In order to demonstrate validity of this lemma, we need to show first that if C_G is a clause constructed in Algorithm ClGrRed for the graph G then for any graph $H = (V, E')$ and the respective associated clause C_H where $E' \subseteq E$, it holds $C_G \triangleleft_* C_H$ if and only if for any pair of vertices v, w connected by a directed walk from v to w in G there is also a directed walk from v to w in H .

(\Rightarrow) For all vertices v and w , if $C_G \triangleleft_* C_H$ and if there is a directed walk from v to w in the graph G then there is also a directed walk from v to w in H : This can be shown easily by inspecting the condition on line 10 of Algorithm \triangleleft_* and from the fact that all vertices in the graph constructed in Algorithm ClGrRed have loops (and therefore if there is a walk of length n connecting given vertices then there is also a walk between these vertices of length m for any $m > n$).

(\Leftarrow) If there is a directed walk in the graph H for any pair of vertices v and w connected by a directed walk in the graph G then $C_G \triangleleft_* C_H$: This follows almost immediately from inspection of line 10 of Algorithm \triangleleft_* . The only thing that might need further justification is that it does not matter that walks of not all lengths are checked there. Only walks of lengths equal to the number of vertices in G and H minus 1 (i.e. $|V| - 1$ because both C_G and C_H contain $|V| - 1$ literals based on the predicate symbol $c/2$). But clearly, if there is a directed walk between two vertices in a graph then there must be a walk between these vertices of length at most equal to the number of vertices in that graph. Moreover, since all vertices in the graph constructed in Algorithm ClGrRed have loops, if there is a walk in the graph of length m where $m < |V|$ then there must also be a directed walk of length $|V|$.

We are almost done. The reduced clause must contain all the constants contained in the original clause (if we removed a constant then there would no longer be any oriented walk from the vertex corresponding to the removed constant to itself). Moreover it must hold $C \triangleleft_* C_{red}$ and therefore the graph corresponding to C_{red} must contain directed walks between pairs of vertices which are also connected in the original graph, as we have shown above.

It remains to show that the set of edges of the resulting graph has minimum cardinality. However, this follows easily from minimality of the clause C_{red} . The only literals which can be removed are the edge/2 literals which do not correspond to loops, therefore the resulting graph must have minimum number of edges among the graphs *equivalent* to G . \square

Proposition 40. *A polynomial time decidable χ -presubsumption exists w.r.t. which finding a maximally reduced clause is NP-hard.*

Proof. Follows from the fact that the procedure for converting the minimum equivalent digraph problem to the problem of finding a maximally reduced clause (used in Algorithm CIGrRed) can be performed in polynomial time, next from Lemma 13 and from Lemma 14. \square

Reducing the complexity of input data is often beneficial for learning. In attribute-value learning, a wide range of *feature selection* methods is available [80]. These methods try to select a strict subset of the original example features (attributes) while maintaining or even improving the performance of the model learned from it with respect to that learned from the original feature set. For binary classification tasks with Boolean features, the REDUCE [77] algorithm has been proposed that removes so called *irrelevant* features. For any model learned with the original feature set, a model with same or better fit on the learning examples may be expressed without the irrelevant features. In the later work [2], the REFER algorithm extended REDUCE to the multiple-class learning setting.

In relational learning, examples are not expressed as tuples of feature values but rather take the form of logical constructs such as first-order clauses. Feature-selection methods are thus not applicable to simplify such learning examples. Here we are interested to see whether also first-order clausal examples can somehow be reduced while guaranteeing that the set of logical formulas which can be induced from such reductions would not be affected.

An obvious approach would be to look for θ -reductions [97] of the input clauses. A θ -reduction of a clause is a smaller, but subsumption-equivalent (and thus also logically equivalent) clause. We have explored an approach based on θ -subsumption before in [61], achieving learning speed-up factors up to 2.63. However, the main problem of θ -reduction is that finding it is an NP-hard problem, rendering the approach practically infeasible in domains with large examples such as those describing protein structures [90].

Here we follow the key idea that the complexity curse can be avoided by sacrificing part of the generality of θ -reduction. In particular, we will look for reductions which may not be equivalent to the original example in the logical sense, but which are equivalent *given the language bias* of the learning algorithm. In other words, if the learning algorithm is not able to produce a hypothesis covering the original example but not covering its reduction (or vice versa), the latter two may be deemed equivalent. For instance, consider the clausal example $\leftarrow \text{atom}(a1) \wedge \text{carbon}(a1) \wedge \text{bond}(a1, a2) \wedge \dots$, whose entire structure is shown in the left of Figure 10. Assume that all terms in hypotheses are variables and hypotheses must have *treewidth* at most 1 or be *acyclic*. Then the learning example is equivalent to the simpler one shown in the right of the figure. Here, we connect the framework of bounded subsumptions and reductions introduced in the previous chapter with reduction of learning examples.

This chapter is structured as follows. We first explain how learning examples can be reduced when hypotheses are guaranteed to be sets of mutually non-resolving clauses in Section 8.1. Then we show that these methods can be used also when the clauses in hypotheses can be mutually resolving in Section 8.2. We evaluate the method experimentally in Section 8.3. Finally, we conclude this chapter in Section 8.4.

8.1 SAFE REDUCTION OF LEARNING EXAMPLES FOR MUTUALLY NON-RESOLVING CLAUSAL THEORIES

The learning task that we consider in this chapter is fairly standard. We are given labelled learning examples encoded as first-order-logic clauses and we would like to find a classifier predicting the class labels of examples as precisely as possible. This task could be solved by numerous relational-learning systems. We aim at finding a reduction procedure that would allow us to reduce the number of literals in the examples while guaranteeing that the coverage of any hypothesis from a pre-fixed hypothesis language \mathcal{L} would not be changed.

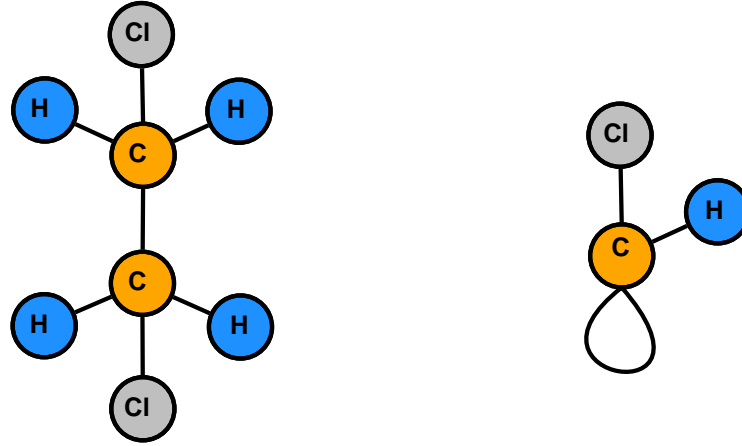


Figure 10: A learning example and its reduction.

There are several settings for logic-based relational learning. We will work within the *learning from entailment setting* [22] which was discussed in Chapter 3. In the learning from entailment, we say that a hypothesis H (a clausal theory) covers an example e (a clause) if and only if $H \models e$. The basic learning task is to find a clausal theory \mathcal{H} that covers all positive examples and no negative examples and contains as few clauses as possible.

Definition 42 (Safe Equivalence and Safe Reduction under Entailment). *Let e and \hat{e} be two clauses and let \mathcal{L} be a language specifying all possible hypotheses. Then \hat{e} is said to be safely equivalent to e if and only if $\forall \mathcal{H} \in \mathcal{L} : (\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \hat{e})$. If e and \hat{e} are safely equivalent and $|\hat{e}| < |e|$ then \hat{e} is called safe reduction of e .*

Clearly, if we have a hypothesis $\mathcal{H} \in \mathcal{L}$ which splits the examples to two sets X and Y then this hypothesis \mathcal{H} will also split the respective set of safely reduced examples to the sets \hat{X}, \hat{Y} containing the safely reduced examples from the sets X and Y , respectively. Also, when predicting classes of test-set examples, any deterministic classifier that bases its decisions on the queries using the covering relation \models will return the same classification even if we replace some of the examples by their safe reductions. The same is also true for propositionalization approaches that use the \models relation to construct boolean vectors which are then processed by attribute-value-learners.

In this chapter, we focus on two types of hypothesis languages: *mutually non-resolving* clausal theories and clausal theories in general. We start with the former type. Recall that we do not put any restrictions on the learning examples. The only restrictions are those put on hypotheses. A *mutually non-resolving* clausal theory is a set of clauses such that no predicate symbol which appears in the head of a clause appears also in the body of any clause. The main reason why we start with non-resolving clausal theories is that logical entailment $\mathcal{H} \models A$, for a non-resolving clausal theory \mathcal{H} and a clause A , can be checked using θ -subsumption. If there is a clause $H \in \mathcal{H}$ such that $H \preceq_{\theta} A$ then $\mathcal{H} \models A$, otherwise $\mathcal{H} \not\models A$.

Bounded equivalence (denoted by \approx_X) introduced in Chapter 7 can be used to check if two learning examples e and \hat{e} are equivalent w.r.t. hypotheses from a fixed hypothesis language. It can be therefore used to search for safe reductions of learning examples. This is formalized in the next proposition. Note that this proposition does not say that e and \hat{e} are equivalent. It merely says that they are equivalent when being used as learning examples in the *learning from entailment* setting with hypotheses drawn from a fixed set.

Proposition 41. *Let \mathcal{L} be a hypothesis language containing only non-resolving clausal theories composed of clauses from a set X and let \triangleleft_X be an χ -presubsumption w.r.t. the set X . If e and \hat{e} are learning examples (not necessarily from X), $e \triangleleft_X \hat{e}$ and $\hat{e} \triangleleft_X e$ then for any $\mathcal{H} \in \mathcal{L}$ it holds $(\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \hat{e})$. Moreover, if $|\hat{e}| < |e|$ then \hat{e} is a safe reduction of e under entailment.*

We now describe several simple safe-reduction transformation methods. For the first transformation method, we assume to have a fixed hypothesis language $\mathcal{L}_{\mathcal{U}}$ consisting of non-resolving clausal theories which contain only constants from a given set \mathcal{U} . The transformation then gets a clause A on its input and produces a new clause \tilde{A} by *variabilizing* constants in A which are not contained in \mathcal{U} . It is easy to check that for any such A and \tilde{A} it must hold $A \approx_{\chi} \tilde{A}$ w.r.t. the set of clauses containing only constants from \mathcal{U} . Therefore A and \tilde{A} are safely equivalent w.r.t. \mathcal{L} . We can think of the constants not used in a hypothesis language \mathcal{L} as identifiers of objects whose exact identity is not interesting for us. Such constants can appear e.g. when we describe molecules and we want to give names to atoms in the molecules with no actual meaning.

Another simple transformation which produces safely equivalent clauses is based on θ -reduction. In this case the set of clauses X can be arbitrary. The transformation gets a clause A on its input and returns its θ -reduction. The χ -equivalence of the clause A and its θ -reduction follows from the fact that θ -subsumption is the χ -subsumption w.r.t. the set of all clauses.

Example 38. Let us have an example

$$e = \text{edge}(a, b, 1) \vee \text{edge}(b, a, 2) \vee \text{edge}(b, c, 2) \vee \text{edge}(c, d, 1) \vee \text{edge}(d, a, 2)$$

and a hypothesis language \mathcal{L} containing arbitrary non-resolving clausal theories with the set of allowed constants $\mathcal{U} = \{1, 2\}$. We variabilize e and obtain a clause

$$\tilde{e} = \text{edge}(A, B, 1) \vee \text{edge}(B, A, 2) \vee \text{edge}(B, C, 2) \vee \text{edge}(C, D, 1) \vee \text{edge}(D, A, 2).$$

Now, e and \tilde{e} are safely equivalent w.r.t. to hypotheses from \mathcal{L} . Next, we obtain a safe reduction of e by computing θ -reduction of \tilde{e} which is $\hat{e} = \text{edge}(A, B, 1) \vee \text{edge}(B, A, 2)$.

Similarly to the example above, if the hypothesis language consists of clausal theories composed of clauses with treewidth at most k then we can reduce the learning examples by the polynomial-time literal-elimination algorithm with the χ -presubsumption based on k -consistency. Importantly, transformations which produce χ -equivalent clauses w.r.t. a set X can be also chained due to transitivity of χ -subsumption. So, for example, if we have a hypothesis language consisting of non-resolving clausal theories which contain only constants from a pre-fixed set \mathcal{U} and we want to safely reduce a clause A then we can first variabilize it and then reduce it using θ -reduction. Moreover, if we know that the clauses in the hypotheses are not only bound to contain just constants from the set \mathcal{U} but we also know that they have treewidth at most k then we can use the literal-elimination algorithm based on k -consistency to reduce the given clause.

8.2 SAFE REDUCTION OF LEARNING EXAMPLES FOR MUTUALLY RESOLVING CLAUSES

In this section, we generalize the methods for reduction of learning examples to hypothesis languages in the form of clausal theories without the restriction that they have to be mutually non-resolving. We start by generalizing the result about variabilization of learning examples, i.e. that by replacing constants which do not appear in the given hypothesis language \mathcal{L} we always obtain an example safely equivalent to the original example w.r.t. \mathcal{L} .

The next proposition shows that we can first variabilize constants forbidden by the hypothesis language \mathcal{L} and then reduce the resulting clause using θ -reduction in order to obtain a safe reduction.

Proposition 42. *Let \mathcal{L} be a hypothesis language and let e be a clause. Let \tilde{e} be a clause obtained from e by variabilizing the constants which are not contained in the hypothesis language. Then $(\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \tilde{e})$ for any $\mathcal{H} \in \mathcal{L}$. If \hat{e} is a θ -reduction of \tilde{e} and $|\hat{e}| < |\tilde{e}|$ then \hat{e} is also a safe reduction of e .*

Note that the above proposition holds for the conventional entailment relation \models . For instance, it would not hold if we used *negation as failure* (because then we could create a $\text{nonequal}(X, Y)$ relation without ever having to use any constant).

We can also generalize the method based on χ -subsumption. This is not as easy as in the case of mutually non-resolving clauses because there are sets of clauses X w.r.t. which χ -reduction could not be used for reduction of learning examples if mutually resolving clausal theories were allowed. Nevertheless, the next proposition gives us a hint how to pick the right sets X for which this is possible.

Proposition 43. *Let X be a set of clauses and let $\mathcal{L} \subseteq 2^X$ be a hypothesis language. Let A and B be clauses. Let $A \approx_X B$ w.r.t. the set X and let the following be true for any $\mathcal{H} \in \mathcal{L}$ and any clause C : if $\mathcal{H} \models C$ and C is not a tautology then there is a clause $D \in X$ such that $\mathcal{H} \models D$ and $D \preceq_\theta C$. Then for any $\mathcal{H} \in \mathcal{L}$, it holds $(\mathcal{H} \models A) \Leftrightarrow (\mathcal{H} \models B)$.*

A trivial example of a hypothesis class which satisfies the conditions from this proposition is the class of mutually non-resolving clausal theories composed from clauses from a given set X . Another trivial example is the class of all hypotheses composed from unrestricted clauses. A disadvantage of the first class is that it is not very rich and a disadvantage of the second class is that it is too rich (and therefore one has to use the NP-hard θ -reduction for safe reduction of examples w.r.t. this class).

Ideally, we would like to find a set of clauses X such that if $\mathcal{L} = 2^X$ then it is possible to safely reduce any example by χ -reduction w.r.t. the set X . The question is how to find such a set. The next proposition gives one possible way for finding such sets. It is based on the subsumption theorem [92] which was briefly reviewed in Chapter 3.

Proposition 44. *Let X be a set of clauses (Horn clauses, respectively) such that any clause which can be derived from a clausal theory $\mathcal{H} \in 2^X$ using resolution (SLD resolution, respectively) is contained in X . If e and \hat{e} are two clauses (Horn clauses, respectively) such that $e \approx_X \hat{e}$ then for any $\mathcal{H} \in 2^X$: $(\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \hat{e})$.*

Next, we show that the conditions from Proposition 44 are satisfied by the important case of Horn clauses with bounded treewidth.

Proposition 45. *Let X_k be the set of all function-free Horn clauses with treewidth at most k . Then for any clause C derivable by SLD resolution from a clausal theory $\mathcal{H} \in 2^{X_k}$ it holds $C \in X_k$.*

Thus, the most important result in this section can be phrased as follows: The safe reduction method based on the χ -reduction w.r.t. the set of bounded-treewidth clauses can be used also when considering theories composed of possibly mutually resolving bounded-treewidth Horn clauses. When learning clausal theories consisting of mutually resolving bounded-treewidth clauses, it is possible to start by reducing learning examples by the literal-elimination algorithm with the polynomial-time χ -presubsumption based on the k -consistency algorithm.

8.3 EXPERIMENTAL EVALUATION OF SAFE REDUCTION

We experimentally evaluate usefulness of the *safe reduction of learning examples* with real-world datasets and two relational learning systems – the popular system Aleph and the state-of-the-art system nFOIL [74]. We implemented *literal-elimination* and *literal-substitution* algorithms for tree-width 1, i.e. for treelike clausal theories. We used the efficient algorithm AC-3 [81] for checking 1-consistency¹. We forced nFOIL and Aleph to construct only clauses with treewidth 1 using their *mode declaration* mechanisms. We used three datasets in the experiments: *predictive toxicology challenge* [46], *CAD* [139] and *hexose-binding proteins* [90]. The PTC dataset contains descriptions of 344 molecules classified according to their toxicity for male rats. The molecules are described using only *atom* and *bond* information. The CAD dataset contains descriptions of 96 class-labelled product-structure designs. Finally, the hexose-binding dataset contains 80 hexose-binding and 80 non-hexose-binding protein domains. Following [90] we represent the protein

¹ Note again the terminology used in this chapter following [4]. In CSP-literature, it is often common to call 2-consistency what we call 1-consistency.

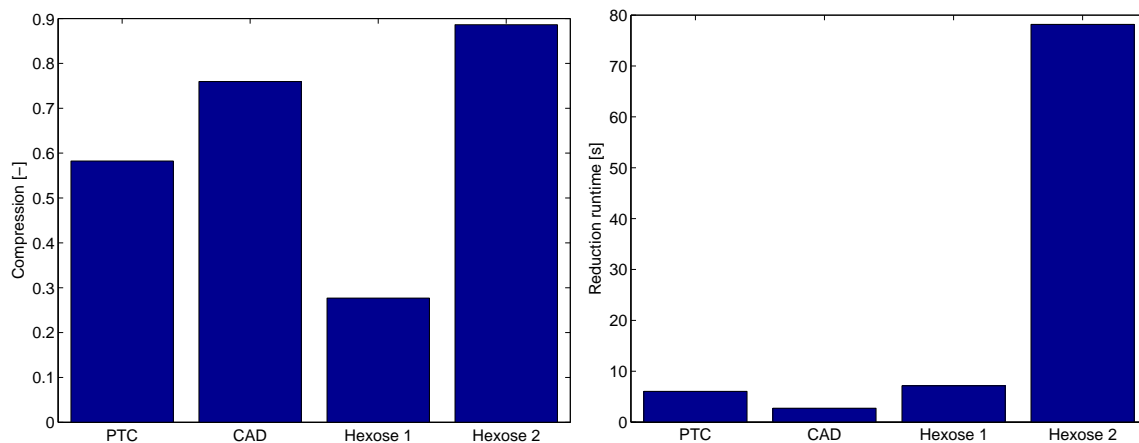


Figure 11: **Left:** Compression rates achieved by *literal-substitution algorithm* on four datasets (for tree-width 1). **Right:** Time for computing reductions of learning examples on four datasets (for treewidth 1).

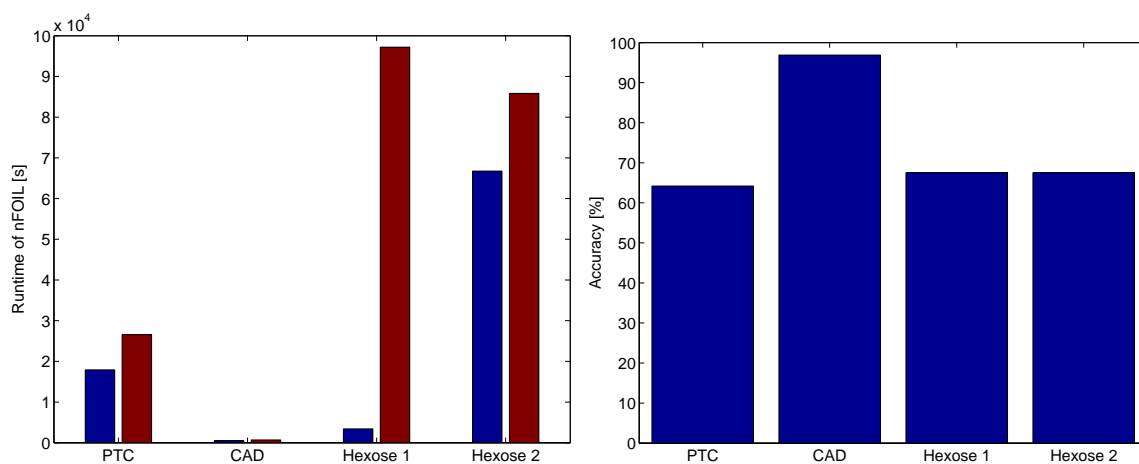


Figure 12: **Left:** Runtime of nFOIL on reduced (blue) and non-reduced (red) datasets. **Right:** Predictive accuracies of nFOIL on four datasets estimated by 10-fold cross-validation.

domains by atom-types and atom-names (each atom in an amino acid has a unique name) and pair-wise distances between the atoms which are closer to each other than some threshold value. We performed two experiments with the last mentioned dataset for cut-off set to 1 Angstrom (Hexose ver. 1) and 2 Angstroms (Hexose ver. 2).

We applied the *literal-elimination* algorithm followed by the *literal-substitution* algorithm on the three datasets. The compression rates (i.e. ratios of number of literals in the reduced learning examples divided by the number of literals in the original non-reduced examples) are shown in the left panel of Figure 11. The right panel of Figure 11 then shows the time needed to run the reduction algorithms on the respective datasets. We note that these times are generally negligible compared to runtimes of nFOIL and with the exception of Hexose ver. 2 also to runtimes of Aleph.

8.3.1 Experiments with nFOIL

We used nFOIL to learn predictive models and evaluated them using 10-fold cross-validation. For all experiments with the exception of the hexose-binding dataset with cut-off value 2 Angstroms, where we used beam-size 50, we used beam-size 100. From one point of view, this is much higher than the beam-sizes used by [74], but on the other hand, we have the experience that this allows nFOIL to find theories which involve longer clauses and at the same

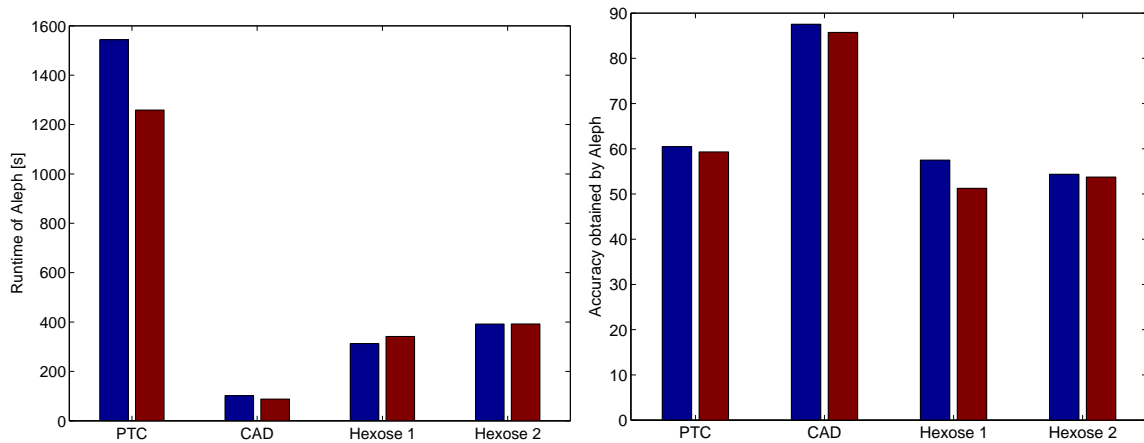


Figure 13: **Left:** Runtime of Aleph on reduced (blue) and non-reduced (red) datasets. **Right:** Predictive accuracies of Aleph on reduced (blue) and non-reduced (red) datasets estimated by 10-fold cross-validation.

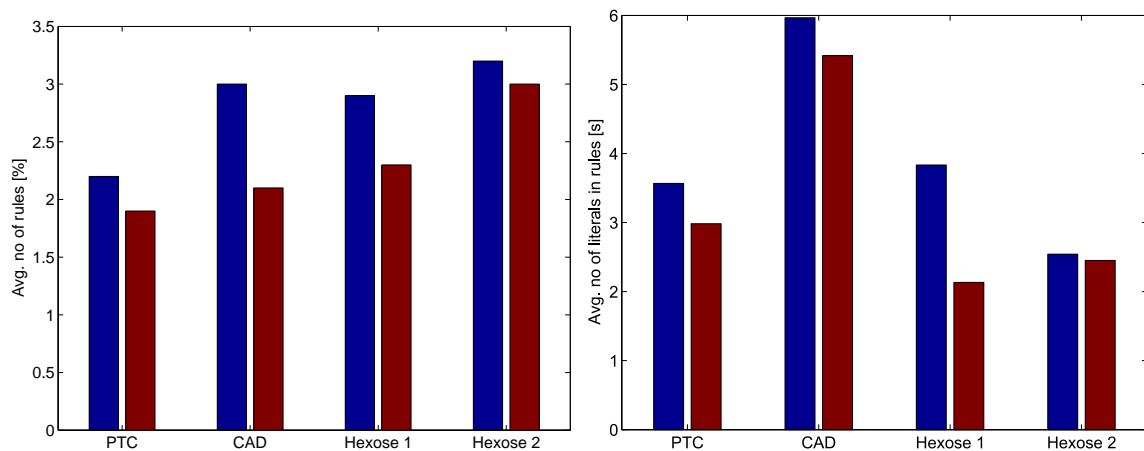


Figure 14: **Left:** Average number of rules generated by Aleph on reduced (blue) and non-reduced (red) datasets. **Right:** Average number of literals in rules generated by Aleph on reduced (blue) and non-reduced (red) datasets.

time have higher predictive accuracies. The runtimes of nFOIL operating on reduced and non-reduced data are shown in the left panel of Figure 12. It can be seen that the reduction was beneficial in all cases but that the most significant speed-up of more than an order of magnitude was achieved on Hexose data. This could be attributed to the fact that nFOIL constructed long clauses on this dataset and the covering test used by it had not probably been optimized. So, in principle, nFOIL could be made faster by optimizing the efficiency of its covering test. The main point, however, is that we can speed-up the learning process for almost any relational learning algorithm merely by preprocessing its input. The right panel of Figure 12 shows nFOIL's predictive accuracies (estimated by 10-fold cross-validation). The accuracies were not affected by the reductions. The reason is that (unlike Aleph) nFOIL exploits learning examples only through the entailment queries.

8.3.2 Experiments with Aleph

We performed another set of experiments using the relational learning system Aleph. Aleph restricts its search space by bottom-clauses. After constructing a bottom-clause it searches for hypotheses by enumerating subsets of literals of the bottom-clause. When we reduce learning examples, which also means reduction of bottom-clauses, we are effectively reducing the size of Aleph's search space. This means that Aleph can construct longer clauses earlier than if it

used non-reduced examples. On the other hand, this also implies that, with the same settings, Aleph may run longer on reduced data than on non-reduced data. That is because computing coverage of longer hypotheses is more time-consuming. Theories involving longer clauses may often lead to more accurate predictions. For these reasons, we measured not only runtime and accuracy, but also the average number of learnt rules and the average number of literals in these rules on reduced and non-reduced data.

We ran Aleph on reduced and non-reduced versions of the datasets and evaluated it using 10-fold cross-validation. We used the literal elimination algorithm for reducing examples. We set the maximum number of explored *nodes* to 50000, the *noise* parameter to 1% of the number of examples in the respective datasets. The runtime in the performed experiments was higher for reduced versions of datasets PTC and CAD, the same for Hexose ver. 2 and lower for Hexose ver. 1 than for their non-reduced counterparts (see left panel of Figure 13). The accuracies were higher for reduced versions of all four datasets (see right panel of Figure 13). Similarly, the average number of rules, as well as the average number of literals in the rules, was higher for the reduced versions of all four datasets (see Figure 14). These results confirm the expectation that Aleph should be able to construct longer hypotheses on reduced datasets which, in turn, should result in higher predictive accuracies.

8.4 CONCLUSIONS

We have introduced a novel concept called *safe reduction*. We have shown how it can be used to safely reduce learning examples (without affecting learnability) which makes it possible to speed-up many relational learning systems by merely preprocessing their input. The methods that we have introduced run in polynomial time for hypothesis languages composed of clauses with treewidth bounded by a fixed constant.

8.5 PROOFS

Proposition 41. *Let \mathcal{L} be a hypothesis language containing only non-resolving clausal theories composed of clauses from a set X and let \triangleleft_X be an χ -presubsumption w.r.t. the set X . If e and \hat{e} are learning examples (not necessarily from X), $e \triangleleft_X \hat{e}$ and $\hat{e} \triangleleft_X e$ then for any $\mathcal{H} \in \mathcal{L}$ it holds $(\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \hat{e})$. Moreover, if $|\hat{e}| < |e|$ then \hat{e} is a safe reduction of e under entailment.*

Proof. First, $e \triangleleft_X \hat{e}$ and $\hat{e} \triangleleft_X e$ imply $e \approx_X \hat{e}$ (where \approx_X denotes χ -equivalence w.r.t. the set X). Then for any non-resolving clausal theory $\mathcal{H} \in \mathcal{L}$ we have $(\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \hat{e})$ because for any clause $A \in X$ we have $(A \preceq_\theta e) \Leftrightarrow (A \preceq_\theta \hat{e})$ (from $e \approx_X \hat{e}$). This together with $|\hat{e}| < |e|$ means that \hat{e} is a safe reduction of e under entailment w.r.t. hypothesis language \mathcal{L} . \square

Lemma 15 (Plotkin [97]). *Let A and B be clauses. If $A \preceq_\theta B$ then $A \models B$.*

Proposition 42. *Let \mathcal{L} be a hypothesis language and let e be a clause. Let \tilde{e} be a clause obtained from e by variabilizing the constants which are not contained in the hypothesis language. Then $(\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \tilde{e})$ for any $\mathcal{H} \in \mathcal{L}$. If \hat{e} is a θ -reduction of \tilde{e} and $|\hat{e}| < |e|$ then \hat{e} is also a safe reduction of e .*

Proof. We will start by showing validity of the implication $(\mathcal{H} \models e) \Rightarrow (\mathcal{H} \models \tilde{e})$. For contradiction, let us assume that there is a model M of the clausal theory \mathcal{H} such that $M \models e$ and $M \not\models \tilde{e}$. Then there must be a substitution θ grounding all variables in \tilde{e} such that² $M \models e\theta$ and $M \not\models \tilde{e}\theta$. Now, we will construct another model M' of \mathcal{H} in which e will not be satisfied. We take each constant c in e that has been replaced by a variable V in \tilde{e} and update the assignment ϕ of the constants c to objects from the domain of discourse in the model³ so that $\phi(c) = \phi(V\theta)$. Clearly, we can do this for every constant c since every constant in e has been replaced by exactly one

² We are applying θ also to e because e need not be ground.

³ A first-order interpretation consists of several components, one of them is the *domain of discourse*, and another is a function ϕ which maps constants to elements of the domain of discourse.

variable. Now, we clearly see that $M' \not\models e$. However, we are not done yet as it might happen that the new model with the modified ϕ would no longer be a model of H . However, this is clearly not the case since none of the constants c appears in H and therefore the change of ϕ has no effect whatsoever on whether or not H is true in M' . So, we have arrived at a contradiction. We have a model M' such that $M' \models H$ and $M' \not\models e$ which contradicts the assumption $H \models e$. The implication $(H \models e) \Leftrightarrow (H \models \tilde{e})$ follows directly from Lemma 15. We have $\tilde{e} \preceq_{\theta} e$ therefore also $\tilde{e} \models e$ and finally $H \models \tilde{e} \models e$. (ii) In order to show $(H \models e) \Rightarrow (H \models \hat{e})$, it suffices to notice that $H \models e$ and $e \preceq_{\theta} \hat{e}$ imply $H \models \hat{e}$. The implication $(H \models e) \Rightarrow (H \models \hat{e})$ may be shown similarly as follows: $H \models \hat{e}$ and $\hat{e} \preceq_{\theta} e$ imply $H \models e$. \square

Proposition 43. *Let X be a set of clauses and let $\mathcal{L} \subseteq 2^X$ be a hypothesis language. Let A and B be clauses. Let $A \approx_X B$ w.r.t. the set X and let the following be true for any $\mathcal{H} \in \mathcal{L}$ and any clause C : if $\mathcal{H} \models C$ and C is not a tautology then there is a clause $D \in X$ such that $\mathcal{H} \models D$ and $D \preceq_{\theta} C$. Then for any $\mathcal{H} \in \mathcal{L}$, it holds $(\mathcal{H} \models A) \Leftrightarrow (\mathcal{H} \models B)$.*

Proof. First, we need to consider the case when A and B are both tautologies. If both A and B are tautologies then $(\mathcal{H} \models A) \Leftrightarrow (\mathcal{H} \models B)$ naturally holds for any \mathcal{H} . Now, we can consider the case when at most one of the clauses A and B is a tautology. Let us assume w.l.o.g. that if one of the clauses is a tautology then it is the clause B . If $\mathcal{H} \models A$ then there is a clause $D \in X$ such that $\mathcal{H} \models D$ and $D \preceq_{\theta} A$ (by the assumptions of the proposition). Since $D \in X$, $D \preceq_{\theta} A$ and $A \preceq_X B$, we have $D \preceq_{\theta} B$ (from the definition of x -subsumption) and finally also $\mathcal{H} \models D \models B$ and so $\mathcal{H} \models B$. The other implication can be shown in a completely similar fashion if A is not a tautology. If A is a tautology then the situation is even simpler because the implication is always true. \square

Proposition 44. *Let X be a set of clauses (Horn clauses, respectively) such that any clause which can be derived from a clausal theory $\mathcal{H} \in 2^X$ using resolution (SLD resolution, respectively) is contained in X . If e and \hat{e} are two clauses (Horn clauses, respectively) such that $e \approx_X \hat{e}$ then for any $\mathcal{H} \in 2^X$: $(\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \hat{e})$.*

Proof. We will use the subsumption theorem (the subsumption theorem for Horn clauses, respectively) and Proposition 43. We will show that the conditions of this proposition imply conditions of Proposition 43. If A is a non-tautological clause and $\mathcal{H} \models A$ then by the subsumption theorem there must be a clause C derivable from \mathcal{H} using resolution (SLD resolution, respectively) such that $C \preceq_{\theta} A$. Therefore for any non-tautological clause A , if $\mathcal{H} \models A$ where $\mathcal{H} \in 2^X$ then there must be a clause $C \in X$ such that $\mathcal{H} \models C$ (because resolution is sound) and $C \preceq_{\theta} A$. Now, since $e \approx_X \hat{e}$, we may finish the proof using Proposition 43 which gives us $(\mathcal{H} \models e) \Leftrightarrow (\mathcal{H} \models \hat{e})$ for any $\mathcal{H} \in 2^X$. \square

Lemma 16 (Clique containment lemma [10]). *Let A be a clause and T_A be its tree decomposition. For any $l \in A$, there is a vertex in T_A labelled by a set of variables V such that $\text{vars}(l) \subseteq V$.*

Proof. The proof can be found in [10]. More precisely, the paper [10] contains a lemma which states that if C is a clique in a graph $G = (V, E)$ then any tree decomposition of G contains a vertex labelled by a set of vertices L such that $C \subseteq L$. Our statement of this lemma in terms of clauses and tree decompositions of clauses then follows immediately from this result which can be shown by noticing that the decomposition of the clause A can be easily converted to a tree decomposition of A 's Gaifman graph G_A where, for any $l \in A$, $\text{var}(l)$ corresponds to a clique in G_A . \square

Lemma 17. *Let A be a function-free clause and T_A be its tree decomposition. Let $l^* \in A$ be a literal and let θ be a substitution affecting only the variables in l^* , mapping variables to variables or terms (i.e. not to function symbols) and never mapping any variables to elements of the set $\text{vars}(A)$. Then a tree decomposition of $A\theta$ can be obtained by applying the substitution θ on the variables contained in the labels of the tree decomposition T_A and removing constants from these sets if necessary – we denote the new labelled tree by $T_A\theta$. As a consequence, the treewidth of $A\theta$ is never greater than the treewidth of A .*

Proof. If we apply the substitution θ on the labels of the tree decomposition T_A then none of the label-sets associated to the vertices of T_A increases in size (this is in part due to the fact that we do not consider function symbols). Therefore if we are able to show that $T_A\theta$ is a tree decomposition of $A\theta$ then we will automatically get also the result that the treewidth of $A\theta$ is not greater than the treewidth of A . So, let us show that $T_A\theta$ is indeed a tree decomposition of $A\theta$.

- (i) Claim: For every variable $V \in \text{vars}(A\theta)$ there is a vertex of $T_A\theta$ labelled by a set containing V . This is obvious.
- (ii) Claim: For every pair of variables U, V which both appear in a literal $l \in A\theta$, there is a vertex of $T_A\theta$ labelled by a set containing both U and V . There must be a literal $l' \in A$ containing two variables U' and V' such that $U'\theta = U$ and $V'\theta = V$ (because U and V are both contained in a literal). Therefore there must be a vertex t of T_A labelled by a set S_t containing both U' and V' . After applying the substitution θ on the set S_t , we get a set by which some vertex contained in $T_A\theta$ is labelled and it contains both U and V .
- (iii) Claim: For every $V \in \text{vars}(A\theta)$, the set of vertices of $T_A\theta$ labelled by sets containing V forms a connected subgraph of $T_A\theta$. Let us assume (for contradiction) that there is a variable $V \in \text{vars}(A\theta)$ such that the set of vertices of $T_A\theta$ labelled by sets containing V forms a disconnected graph. It follows that there must be two variables U', V' ($U' \neq V'$) such that $U'\theta = V'\theta = V$ and the sets of vertices $S_{U'}$ and $S_{V'}$ of the tree decomposition T_A corresponding to the variables U' and V' , respectively, must be disjoint (because the set of vertices with labels containing a given variable must form a connected subgraph in any tree decomposition). However, the variables U' and V' must appear in the literal l^* because the substitution θ affects only variables contained in l^* and maps no variables to elements of the set $\text{vars}(A)$ (since at least one of the variables must have been affected by the substitution and since it is equal to the other variable, the other variable must have been affected by the substitution as well). Thus, since both U' and V' must be contained in l^* there must be a vertex in the tree decomposition T_A labelled by a set which contains both U' and V' . The sets of vertices of T_A labelled by sets containing the variables U' and V' , respectively, therefore cannot be disjoint which is a contradiction.

We have thus shown that $T_A\theta$ is a tree decomposition of $A\theta$. □

Lemma 18. Let $A = l_1 \vee l_2 \vee \dots \vee l_m$ and $B = m_1 \vee m_2 \vee \dots \vee m_n$ be two standardized-apart function-free clauses. Let θ be a most general unifier of the literals $l_i, \neg m_j$ such that $\text{vars}(l_i\theta) \cap \text{vars}(A) = \text{vars}(l_i\theta) \cap \text{vars}(B) = \emptyset$. Next, let

$$C = (l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_m \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)\theta$$

be a binary resolvent of A and B . Then for the treewidth k_C of C , it holds $k_C \leq \max\{k_A, k_B\}$ where k_A is the treewidth of A and k_B is the treewidth of B .

Proof. Using Lemma 17, we get that $A\theta$ has treewidth at most k_A and $B\theta$ at most k_B . Therefore also (because if $X \subseteq Y$ then X has treewidth lower or equal than Y):

$$A' = (l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_m)\theta$$

has a tree decomposition $T_{A'}$ of width at most k_A and

$$B' = (m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)\theta$$

has a tree decomposition $T_{B'}$ of width at most k_B . We will now show how to construct a tree decomposition of width at most $\max\{k_A, k_B\}$ for the clause C . Let V_A (V_B , respectively) be a vertex from $T_{A'}$ ($T_{B'}$, respectively) which is labelled by a set of variables \mathcal{V}_A (\mathcal{V}_B , respectively) such that $\text{vars}(l_i\theta) \subseteq \mathcal{V}_A$ ($\text{vars}(l_i\theta) \subseteq \mathcal{V}_B$) – such vertices must exist by Lemma 16. We construct

the new tree decomposition T_C by connecting $T_{A'}$ and $T_{B'}$ by a new edge between the vertices V_A and V_B . Clearly, T_C has width at most $\max\{k_A, k_B\}$. We need to show that it is indeed a tree decomposition of C . The first two conditions from Definition 11 are trivially satisfied which follows from the fact that $T_{A'}$ and $T_{B'}$ are tree decompositions of the two clauses A' and B' from which C is composed ($C = A' \cup B'$). It remains to show validity of the third condition (connectedness).

Let us assume (for contradiction) that there is a variable $V \in \text{vars}(C)$ such that the vertices labelled by sets containing the variable V form a disconnected subgraph of T_C . The variable V cannot be contained in $\text{vars}(A)$ or $\text{vars}(B)$. If $V \in \text{vars}(A)$ then $V \notin \text{vars}(B)$ (because A and B were standardized apart) and also $V \notin \text{vars}(l_i\theta)$ (because we selected the unifier θ to satisfy $\text{vars}(l_i\theta) \cap \text{vars}(A) = \emptyset$) but then the set of vertices labelled by sets containing the variable V could not be disconnected because it is actually connected in T_A and none of the labels in T_B contains V . The same argument can be used to show that $V \notin \text{vars}(B)$. So the only remaining possibility is that $V \in \text{vars}(l_i\theta)$. However, this is not possible either. Since both $T_{A'}$ and $T_{B'}$ are tree decompositions, the set of vertices labelled by the sets containing the variable V forms a connected subgraph in both $T_{A'}$ and $T_{B'}$. Moreover, a vertex from $T_{A'}$ and a vertex from $T_{B'}$ which are both labelled by sets containing all variables from $\text{vars}(l\theta)$ are connected by an edge in T_C therefore the set of vertices labelled by the sets containing the variable V must form a connected subgraph of T_C . Thus, we have arrived at a contradiction because there cannot be any variable with a disconnected subgraph of T_C associated to it.

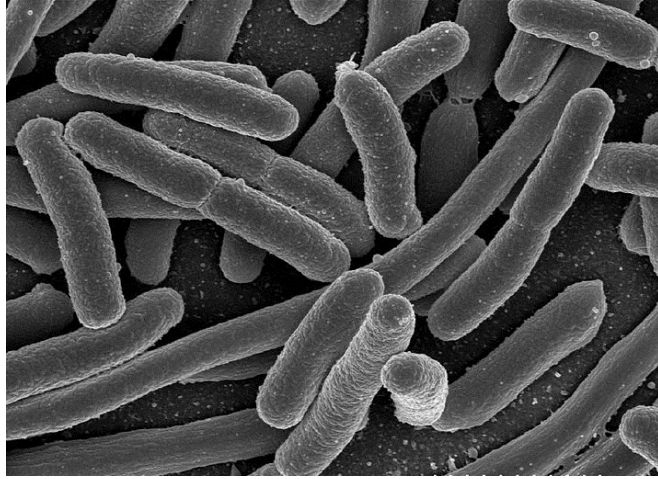
We have verified that T_C is a tree decomposition of C with width at most $\max\{k_A, k_B\}$. \square

Proposition 45. *Let X_k be the set of all function-free Horn clauses with treewidth at most k . Then for any clause C derivable by SLD resolution from a clausal theory $\mathcal{H} \in 2^{X_k}$ it holds $C \in X_k$.*

Proof. Since this proposition considers only SLD resolution, we can consider just the case of binary resolvents (we do not need to take factors into account). The proposition then follows immediately from Lemma 18 because any clause derived by applying the binary resolution rule on two clauses must always have treewidth bounded by the treewidth of the clauses from which it was derived. \square

Part IV

BIOINFORMATICS APPLICATIONS



**Illustration adapted from en.wikipedia.*

Prediction of DNA-binding propensity of proteins is an important biological problem that has not been fully solved yet. Proteins which bind to DNA play various essential roles in living cells, including transcription, replication and maintenance of DNA and regulation of gene expression. Knowing which proteins bind to DNA is important for several reasons. If we know which proteins bind to DNA, we can better understand various processes that occur in living cells on the microscopic level. Being able to detect which proteins bind to DNA and preferably also at which sites is also beneficial for designing new proteins that could be potentially useful for emerging gene therapies. In this chapter we present an approach for prediction of DNA-binding propensity of proteins based on the feature-construction algorithm RelF (described in Chapter 5). Here, RelF is used for construction of features describing local spatial configurations of amino acids in proteins. The present approach is different from some of most recent methods based on similarity of proteins, for example structural alignment or threading-based methods [140, 38, 39], or methods exploiting information about evolutionary conservation of amino acids in proteins [94]. In general, methods exploiting evolutionary information can be more accurate than approaches aiming to infer binding propensity purely from physicochemical or structural protein properties. On the other hand, the main advantage of approaches not using evolutionary information is that since they do not rely on the existence of homologous proteins, they can be used to predict DNA-binding function for proteins with no known homologs.

This chapter is organized as follows. We review the most important existing approaches to DNA-binding propensity prediction in Section 9.1. Then we describe the datasets of proteins used in the experiments presented in this chapter in Section 9.2. We outline the method for prediction of DNA-binding propensity of proteins in Section 9.3. We present results of the experiments in Section 9.4. Section 9.5 concludes this chapter.

9.1 BACKGROUND AND RELATED WORK

In one of the first works on prediction of DNA-binding propensity, Stawiski et al. [114] investigated the structural and sequence properties of large, positively charged electrostatic patches on DNA-binding protein surfaces. They used a neural network with 12 features. Ahmad and Sarai [1] developed another method based on neural networks using the following attributes: the net charge, the electric dipole and quadrupole moments of the protein. The combination of charge and dipole moment turned out to perform well, while information about the quadrupole moment further improved the accuracy only slightly. They found out that DNA-binding proteins have significantly higher net positive charges and electric moments than other proteins. More recently, Szilágyi and Skolnick [120] created a logistic regression classifier based on the amino acid composition, the asymmetry of the spatial distribution of specific residues and the dipole moment. Nimrod et al. [94] presented a random-forest-based method which relies heavily on evolutionary information and works as follows. First, it detects clusters of evolutionarily conserved regions on the surface of proteins using the PatchFinder algorithm. Next, a classifier is trained using features like the electrostatic potential, cluster-based amino acid conservation patterns, the secondary structure content of the patches and features of the whole protein, including all the features used by Szilágyi and Skolnick [120].

Nassif et al. [89] previously used a relational learning based approach in a similar context, in particular to classify hexose-binding proteins. The main differences of our approach from the method of Nassif et al. are as follows. First, the fast relational learning algorithm that we use enables us to produce features by inspecting much larger structures (up to tens of thousands of entries in a learning example) than those considered in the work of Nassif et al. using

the standard learning system Aleph. Second, our structural features acquire values equal to the number of occurrences of the corresponding spatial pattern, whereas Nassif et al. only distinguished the presence of a pattern in a learning example from its absence. Our preliminary results [116] indicated that occurrence-counting indeed substantially lifts predictive accuracy. Lastly, the approach of Nassif et al. resulted in classifiers that are more easily interpretable than state-of-the-art classifiers and comparable in predictive accuracy. Here we maintain the interpretability advantage and achieve accuracies competitive to the state-of-the-art predictive accuracies both by a purely structural approach (without the physicochemical features) and also through the combination of structural and physicochemical features.

9.2 DATASETS

DNA-binding proteins are composed of DNA-binding domains, which are independently folded protein domains that contains at least one motif that recognizes double-stranded or single-stranded DNA. The following datasets of DNA-binding proteins and non-DNA-binding proteins were used in the experiments described in this chapter.

- PD138 - a dataset of 138 DNA-binding protein structures in complex with DNA,
- UD54 - a dataset of 54 DNA-binding protein structures in unbound conformation,
- BD54 - a dataset of 54 DNA-binding protein structures in DNA-bound conformation corresponding to the set UD54
- APO104 - a dataset of 104 DNA-binding protein structures in unbound conformation,
- NB110 - a dataset of 110 non-DNA-binding protein structures,
- NB843 - a dataset of 843 non-DNA-binding protein structures.

The dataset PD138 was created by Szilágyi and Skolnick [120] using the Nucleic Acid Database (NDB). It contains DNA-binding proteins in complex with DNA strands. The maximum pairwise sequence identity of any pair of proteins in this dataset is 35%. Both the protein and the DNA can alter their conformation during the process of binding. This conformational change can involve small changes in side-chain location, and also local refolding, in the case of the proteins. Predicting DNA-binding propensity from a structural model of a protein makes sense if the available structure is not a protein-DNA complex, i.e. if it does not contain a bound nucleic acid molecule. In order to find out how the results would change according to the conformation before and after binding, we used two other datasets (UD54, BD54). The dataset BD54 contains bound conformations of DNA-binding proteins, i.e. DNA-protein complexes. The dataset UD54 contains the same sequences in their unbound, free conformation. These datasets were also obtained from Szilágyi and Skolnick [120]. Another set of DNA-binding protein structures (APO104) determined in the absence of DNA was obtained from Gao et al. [38].

The dataset NB110 contains non-DNA-binding proteins. Originally, Rost and Sander constructed a dataset RS126 which was intended for secondary structure prediction. Ahmad & Sarai [1] then removed the proteins related to DNA binding from it, which gave rise to the dataset of non-DNA-binding proteins NB110. We also used an extended dataset NB843 which was compiled by Nimrod et al. [94]. This dataset contains all proteins from the dataset NB110 and 733 additional structures of non-DNA-binding proteins. These additional structures were gathered using the PISCES server. Entries in this list include crystal structures with a resolution better than 3.0Å. The sequence identity between each pair of sequences is smaller than 25%.

9.3 METHOD

Our method exploits techniques of propositionalization. It uses the feature-construction algorithm RelF in conjunction with state-of-the-art attribute-value learning algorithms. Very briefly,

our method can be viewed as proceeding in three steps. It starts with PDB¹ files which contain descriptions of experimentally determined spatial structures of proteins. Then it creates a relational representation of the proteins. After that it employs RelF to extract meaningful counting relational features from the relational structures describing proteins and uses them to create an approximate attribute-value representation – an attribute-value table where attributes correspond to the automatically discovered features. This attribute-value representation is then used as input to train attribute-value classifiers.

The method uses a relational representation of proteins that consists of literals representing types of amino-acid residues and literals representing pair-wise distances between the residues up to maximum distance 10Å. These distances are computed from alpha-carbon coordinates obtained from PDB². Despite the relative simplicity of our relational representation of proteins, some of the examples contained, in the end, tens of thousands literals which would be very challenging for common relational learning systems such as Aleph [111], not to mention that these systems do not allow computing numbers of covering substitutions. Our representation of proteins differs from the representation used by Nassif et al. [90] in that it describes proteins on the amino-acid level whereas Nassif et al. described their proteins on the atomic level. There are several reasons for devising and using this new representation. First, some proteins in the datasets that we use are not described on the atomic-level but contain only coordinates of alpha-carbon atoms of individual amino-acid residues. Thus, we would have to discard valuable learning examples corresponding to these proteins if we wanted to use the atomic-level description of protein structures. Second, the detailed configurations of atoms in amino acids should be, at least implicitly, determined by the relative positions of the amino-acids' alpha carbons. Moreover, DNA-binding propensity has been predicted with relatively high accuracy using much cruder level of detail (recall for instance the methods based on global physicochemical properties of proteins mentioned in Section 9.1). Therefore it is natural to expect that it should be possible to predict DNA-binding propensity of proteins also from the representation of proteins on the amino-acid level. Third, already the representation on amino-acid level gives rise to learning examples involving tens of thousands of literals and the learning examples would be even bigger for the atom-level representation.

9.4 EXPERIMENTS, RESULTS AND DISCUSSION

We performed experiments with the datasets described in Section 9.2 in order to evaluate the predictive accuracy and also the interpretability of our approach. We compared classifiers based on structural features discovered by our method (SF) with classifiers based on 10 physicochemical features (PF) identified as most predictive by Szilágyi and Skolnick's method [120]. We also experimented with classifiers based on the combination of the structural features and the physicochemical features (PSF). For each experiment we estimated predictive accuracy and the area under the ROC curve (AUC) by 10-fold cross-validation. Lastly, we inspected the most informative, according to χ^2 criterion [79], structural features in order to evaluate their interpretability. We used six state-of-the-art attribute-value learning algorithms listed in Table 14. We used implementation of these learning algorithms present in the WEKA [133] open-source machine learning software.

Parameters of the classifiers were tuned using internal cross-validation. When performing cross-validation, the set of features was created separately for each train-test split corresponding to iterations of cross-validation procedure. The number of trees for random forests and the number of iterations for Ada-boost was selected from the set {10, 20, 50, 100, 200, 500, 1000}. The complexity parameter c for linear support vector machine and for support vector machine with RBF kernel was selected from the set {1, 10, 10², 10³, 10⁴, 10⁵, 10⁶}. The regularization parameter

¹ PDB is a widely used format for protein structures.

² <http://www.pdb.org>

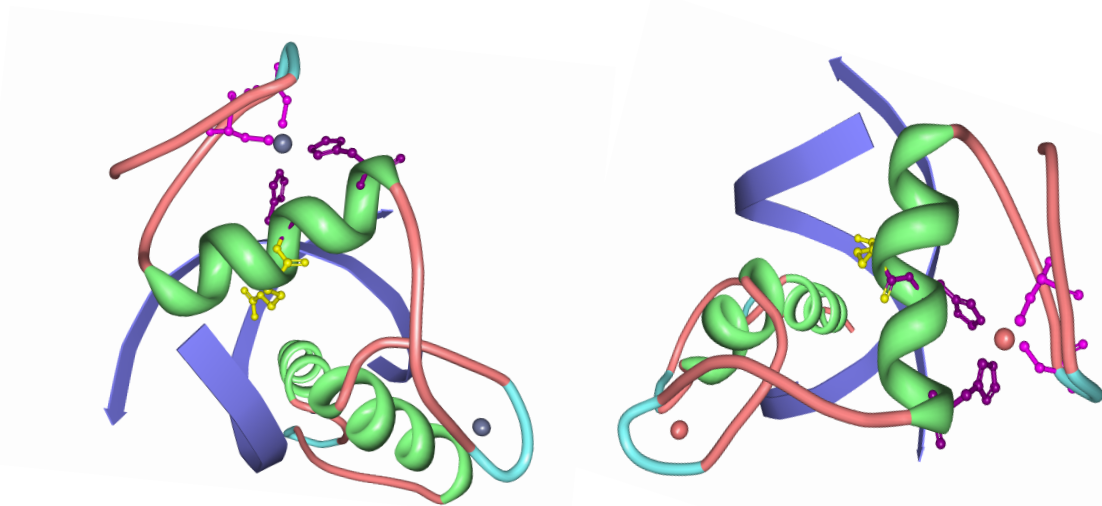


Figure 15: Example zinc-finger proteins (1A1F and 1AAY) containing one discovered relational feature displayed using the protein viewer software [85]. Residues assumed by the pattern are indicated in the following way: CYS - pink, HIS - violet, ARG - yellow.

CLASSIFIER	CATEGORY	REFERENCES
Linear support vector machine	kernel	[15]
SVM with RBF kernel	kernel	[15]
Simple logistic regression	regression/ensemble	[72]
L ₂ -regularized logistic regression	regression	[47]
Ada-boost (with decision stamps)	ensemble	[36]
Random forest	ensemble	[11]

Table 14: State-of-the-art attribute-value learning algorithms used for classification.

of L₂-regularized logistic regression was selected from the set $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$. The minimum frequency of features on one of the classes was set to 0.7.

We used a different methodology for experiments with the dataset PD138/NB843, because the size of this dataset required a sampling-based approach to feature construction rather than exhaustive search. Therefore, we followed an approach in which features were constructed on several randomly selected subsets of data and then evaluated on the complete dataset. The number of random samples was set to 10, the number of proteins in the samples from each class was set to 20. The minimum frequency for each sample was set to 1.

We performed four sets of experiments with datasets of DNA-binding proteins - PD138, UD54, BD54 and APO104 – each one as a set of *positive examples* and the dataset of non-DNA-binding proteins NB110 - as a set of *negative examples*. We obtained about 1400 features for the dataset PD138/NB110, approximately 1500 structural features for the dataset UD54/NB110, about 2400 structural features for the dataset BD54/NB110 and about 2800 structural features for the dataset APO104/ NB110. Accuracies and areas under the ROC curve (AUC) obtained on these datasets by stratified 10-fold cross validation using physicochemical features (PF), structural features (SF) and the combination of both of them (PSF) are shown in Tables 16, 17, 15 and 18. The results for the method based on physicochemical features (PF) differ slightly from the results reported

by Szilágyi and Skolnick [120], because we used 10-fold cross-validation whereas Szilágyi and Skolnick used leave-one-out cross-validation.

We computed average rankings over the six machine learning algorithms for accuracies and AUCs. The method based on purely structural features (SF) and the method based on the combination of structural and physicochemical features (PSF) achieved higher predictive accuracies than the method based purely on physicochemical features (PF). The only exception was in case of the dataset BD54/NB110, where the method based on purely physicochemical features performed better than the method based on purely structural features. The results were not as definite in the case of AUC as in the case of predictive accuracy. The method based on structural features turned out to be better than the method based on physicochemical features on two datasets. Interestingly, these two datasets contain DNA-binding proteins in their unbound conformations. The method based on the combination of structural and physicochemical features was better than the method based on purely physicochemical features on three datasets.

We made an additional experiment with the dataset PD138/NB843 in order to be able to compare our method with the method of Nimrod et al. [94]. In this experiment we used only the random forest classifier which was also used by Nimrod et al. On this dataset Nimrod et al. obtained AUC 0.9. We obtained AUC 0.84 with the method of Szilágyi and Skolnick, 0.82 with the method based on structural features and 0.82 with the method based on the combination of structural and physicochemical features. It is important to note that unlike the method of Nimrod et al. our method does not rely on information about evolutionary conservation.

It is interesting to compare the results for the datasets UD54 and BD54. The dataset UD54 contains DNA-binding proteins in unbound conformation, the dataset BD54 contains the same DNA-binding proteins, but in bound conformation with DNA. Whereas the highest predictive accuracies and best AUCs were obtained by the method based on structural features on the dataset UD54, this method performed worst on the dataset BD54. Interestingly, the number of frequent structural features was significantly higher for the dataset BD54 (approximately 2400 structural features) than for the dataset UD54 (approximately 1500 structural features). This suggests that conformational changes after DNA-binding give rise to greater variability of spatial arrangements of some amino acid groups. Moreover, conformational changes may be responsible for increase of spatial asymmetry of some amino acids or protein's dipole moment. This can explain the better performance of the method based on physicochemical features on the dataset BD54 (recall that these features were selected by experimenting on DNA-binding proteins in bound conformation with DNA by Szilágyi and Skolnick [120]). Also note that prediction of DNA-binding propensity from unbound conformations is more important for practical applications.

9.4.1 *Evaluation of binding motif independence*

In order to find out whether the features discovered for DNA-binding proteins did not just capture the consensus patterns of particular protein folds, we performed an experiment in which the relational learning model was always constructed for proteins from all but one protein group and then tested on this excluded group. Proteins of the dataset PD138 were divided into seven groups following the work of Szilágyi and Skolnick [120]. They were the following: helix-turn-helix, zinc-coordinating, zipper-type, other α -helix, β -sheet, other and enzyme. We used linear SVM based on our structural features (SF), because SVM turned out to perform best in our experiments. We show both the predictive accuracies obtained by testing the learnt classifiers on the excluded groups and the cross-validated accuracies obtained by the classifiers on the remaining parts of the dataset in Table 19. The resulting accuracies on the excluded groups, which should correlate with the ability of our method to discover patterns characteristic for DNA-binding proteins in general, are reasonably high with the exception of the enzyme group. This agrees with the results of Szilágyi and Skolnick [120] and Stawiski et al. [114], who also noticed a drop in the ability of their method to detect DNA-binding proteins in the enzyme group. We can conclude that our method is indeed able to construct classifiers which can work

PD ₁₃₈ vs. NB ₁₁₀	Accuracy			AUC		
	PF	SF	PSF	PF	SF	PSF
Simple log. regr.	83.4 (1)	82.2 (2)	80.7 (3)	0.91 (2)	0.90 (3)	0.94 (1)
L ₂ -reg. log. regr.	81.4 (3)	83.5 (2)	85.5 (1)	0.92 (1)	0.91 (2)	0.91 (2)
SVM with RBF	81.8 (2)	79.9 (3)	85.1 (1)	0.92 (2)	0.90 (3)	0.93 (1)
Linear SVM	81.4 (3)	83.6 (2)	83.9 (1)	0.92 (2)	0.89 (3)	0.93 (1)
Ada-boost	80.6 (2)	78.6 (3)	81.4 (1)	0.90 (1)	0.90 (1)	0.90 (1)
Random forest	81.8 (3)	83.5 (1)	82.3 (2)	0.90 (3)	0.91 (2)	0.93 (1)
Average ranking	2.33	2.17	1.5	1.83	2.33	1.17

Table 15: Predictive accuracies and areas under the ROC curve (*AUC*) on the dataset PD₁₃₈/NB₁₁₀ achieved by 6 machine learning algorithms using physicochemical features (*PF*) as proposed by Szilágyi and Skolnick [120], structural features (*SF*) automatically constructed by our algorithm, and the combination of both feature sets (*PSF*).

UD ₅₄ vs. NB ₁₁₀	Accuracy			AUC		
	PF	SF	PSF	PF	SF	PSF
Simple log. regr.	81.0 (3)	86.0 (1)	82.8 (2)	0.91 (1)	0.89 (2)	0.89 (2)
L ₂ -reg. log. regr.	82.2 (3)	82.4 (2)	84.1 (1)	0.89 (3)	0.91 (1)	0.90 (2)
SVM with RBF	81.0 (2)	84.0 (1)	80.4 (3)	0.92 (1)	0.88 (3)	0.91 (2)
Linear SVM	81.7 (2)	82.4 (1)	82.4 (1)	0.90 (2)	0.91 (1)	0.87 (3)
Ada-boost	76.2 (3)	78.0 (2)	79.3 (1)	0.88 (3)	0.89 (2)	0.90 (1)
Random forest	78.6 (3)	79.3 (1)	79.2 (2)	0.88 (3)	0.89 (2)	0.90 (1)
Average ranking	2.67	1.34	1.67	2.17	1.67	2

Table 16: Predictive accuracies and areas under the ROC curve (*AUC*) on the dataset UD₅₄/NB₁₁₀ achieved by 6 machine learning algorithms using physicochemical features (*PF*) as proposed by Szilágyi and Skolnick [120], structural features (*SF*) automatically constructed by our algorithm, and the combination of both feature sets (*PSF*).

BD ₅₄ vs. NB ₁₁₀	Accuracy			AUC		
	PF	SP	PSP	PF	SP	PSP
Simple log. regr.	80 (3)	80.5 (2)	81.8 (1)	0.91 (1)	0.85 (2)	0.91 (1)
L ₂ -reg. log. regr.	83.1 (1)	81.9 (2)	81.7 (3)	0.92 (1)	0.88 (3)	0.91 (2)
SVM with RBF	82.5 (2)	82.5 (2)	83.6 (1)	0.91 (1)	0.90 (2)	0.90 (2)
Linear SVM	81.4 (3)	82.3 (2)	82.9 (1)	0.93 (2)	0.90 (3)	0.94 (1)
Ada-boost	84.2 (1)	73.8 (3)	79.8 (2)	0.91 (1)	0.88 (2)	0.88 (2)
Random forest	82.4 (1)	75.0 (3)	79.4 (2)	0.89 (2)	0.89 (2)	0.91 (1)
Average ranking	1.83	2.33	1.67	1.33	2.33	1.5

Table 17: Predictive accuracies and areas under the ROC curve (*AUC*) on the dataset BD₅₄/NB₁₁₀ achieved by 6 machine learning algorithms using physicochemical features (*PF*) as proposed by Szilágyi and Skolnick [120], structural features (*SF*) automatically constructed by our algorithm, and the combination of both feature sets (*PSF*).

APO104 vs. NB110	Accuracy			AUC		
	PF	SF	PSF	PF	SF	PSF
Simple log. regr.	80.7 (3)	85.0 (1)	80.8 (2)	0.89 (3)	0.92 (1)	0.91 (2)
L ₂ -reg. log. regr.	82.6 (3)	84.5 (1)	83.1 (2)	0.90 (2)	0.91 (1)	0.91 (1)
SVM with RBF	79.4 (3)	83.2 (2)	84.1 (1)	0.88 (3)	0.90 (2)	0.91 (1)
Linear SVM	79.4 (3)	84.5 (1)	84.1 (2)	0.89 (2)	0.89 (2)	0.92 (1)
Ada-boost	77.6 (3)	78.1 (2)	79.1 (1)	0.87 (2)	0.87 (2)	0.89 (1)
Random forest	81.7 (1)	78.5 (3)	79.4 (2)	0.88 (2)	0.87 (3)	0.89 (1)
Average ranking	2.67	1.67	1.67	2.33	1.83	1.17

Table 18: Predictive accuracies and areas under the ROC curve (*AUC*) on the dataset APO104/NB110 achieved by 6 machine learning algorithms using physicochemical features (*PF*) as proposed by Szilágyi and Skolnick [120], structural features (*SF*) automatically constructed by our algorithm, and the combination of both feature sets (*PSF*).

PROTEIN GROUP	ACC. ON EXCL. GROUP	CV ACC. ON TRAINING DATA
Helix-turn-helix	83.3	80.3
Zinc-coordinating	100	82.9
Zipper-type	88.9	83.1
Other α -helix	100	85.0
β -sheet	77.8	86.0
Other	100	82.5
Enzyme	58.1	90.4

Table 19: Predictive accuracies obtained by linear SVM classifiers trained on the datasets PD138/NB110 with protein groups excluded from PD138. The *accuracy on excluded group* is the percentage of correctly classified proteins from the protein group excluded from the training data. The *cross-validated accuracy on training data* is the accuracy of the learnt model estimated by 10-fold cross-validation on the training data.

accurately over various (non-enzyme) groups of proteins and that its ability to detect DNA-binding proteins is not due to discovery of conserved consensus patterns of different protein folds.

9.5 CONCLUSIONS

We applied relational machine learning techniques to predict DNA-binding propensity of proteins. We utilized the feature construction algorithm RelF described in Chapter 5. We have shown that our relational learning approach is competitive to a state-of-the-art physicochemical approach for DNA-binding propensity prediction in terms of predictive accuracy.

Antimicrobial peptides are molecules responsible for defence against microbial infections in the first stages of the immunological response. Recently antimicrobial peptides have been recognized as a potential replacement of conventional antibiotics for which some microorganisms had already acquired resistance. Although, there are theories about the mechanisms by which antimicrobial peptides kill pathogenic microorganisms, the process has not been fully uncovered yet. Several computational approaches have been developed in past years to predict antimicrobial activity of peptides [16, 124].

In this chapter we describe an approach to prediction of antimicrobial activity from modelled spatial structure information. We utilize the feature-construction algorithm RelF. We use the same relational representation that we used for DNA-binding propensity prediction of proteins in Chapter 9. Whereas RelF was only used for classification problems so far, in this chapter, we use it for regression (prediction of antimicrobial activity, which is a continuous variable). We show that this approach improves on a state-of-the-art method to antimicrobial activity prediction.

10.1 ANTIMICROBIAL PEPTIDES

Antimicrobial peptides (AMPs) have been actively researched for their potential therapeutic application against infectious diseases. AMPs are amino acid sequences of length typically from 6 to 100. They are produced by living organisms of various types as part of their innate immune system [96]. They express potent antimicrobial activity and are able to kill a wide range of microbes. In contrast to conventional antibiotics, AMPs are bacteriocidal (i.e. bacteria killer) instead of bacteriostatic (i.e. bacteria growth inhibitor). Most AMPs work directly against microbes through a mechanism which starts with membrane disruption and subsequent pore formation, allowing efflux of essential ions and nutrients. According to current view this mechanism works as follows: AMPs bind to the cytoplasmic membrane and create micelle-like aggregates, which leads to disruption of the membrane. In addition, there may be complementary mechanisms such as intracellular targeting of cytoplasmic components crucial to proper cellular physiology. Thus, the initial interaction between the peptides and the microbial cell membrane allows the peptides to penetrate into the cell to disrupt vital processes, such as cell wall biosynthesis and DNA, RNA, and protein synthesis. A convenient property of AMPs is their selective toxicity to microbial targets, which makes them non-toxic to mammalian cells. This specificity is based on the significant distinctions between mammalian and microbial cells, such as composition, transmembrane potential, polarization and structural features.

Antimicrobial peptides are small, positively charged, amphipathic molecules. They include two or more positively charged residues and a large proportion of hydrophobic residues. Many AMPs exist in relatively unstructured conformations prior to interaction with target cells. Upon binding to pathogen membranes, peptides may undergo significant conformational changes to helical or other structures. These conformations of antimicrobial peptides may impact their selective toxicity [138]. The three-dimensional folding of the peptides results in the hydrophilic or charged amino acids segregating in space from the hydrophobic residues, leading to either an amphipathic structure, or a structure with two charged regions spatially separated by a hydrophobic segment. Such a structure can interact with the membrane [44]. The amphipathicity of the AMPs enables insertion into the membrane lipid bilayer.

10.2 EXISTING APPROACHES TO ACTIVITY PREDICTION

Several methods have been developed to predict antimicrobial activity of AMPs with potential therapeutic application. Some algorithms take advantage of data mining and high-throughput screening techniques and apply attribute-value approach to scan protein and peptide sequences [75, 123]. Similar strategies were proposed based on supervised learning techniques, such as artificial neural networks or support vector machines, in order to evaluate amounts of complex data [52]. Most attempts have been focused to the prediction of peptide's activity using quantitative structure-activity relationships (QSAR) descriptors together with artificial neural networks [53, 33, 16], linear discriminant [122] or principal component analysis [121]. A QSAR-based artificial neural network system was experimentally validated using SPOT high-throughput peptide synthesis, demonstrating that this methodology can accomplish a reliable prediction [32]. Recently, an artificial neural network approach based on the peptide's physicochemical properties has been introduced [124]. These properties were derived from the peptide sequence and were suggested to comprise a complete set of parameters accurately describing antimicrobial peptides.

The approach that we propose in this chapter differs from the above mentioned approaches mainly in the following. Rather than using an ad-hoc set of physicochemical properties of the peptides, we use an automatic feature construction method based on relational machine learning to discover structural features capturing spatial configuration of amino acids in peptides. A positive aspect of our method (besides improving predictive accuracy) is that it provides us with interpretable features involving spatial configurations of selected amino acids. Moreover, it is not limited to the prediction of antimicrobial activities as it can easily be used also for prediction of other numeric properties of peptides.

10.3 ANTIMICROBIAL ACTIVITY PREDICTION

Our approach to antimicrobial-activity prediction exploits structure prediction methods and techniques of relational machine learning in conjunction with state-of-the-art attribute-value learning algorithms. Very briefly, our method can be imagined as proceeding in four steps. In the first step, 3D structures of peptides are computed using LOMETS software [135]. LOMETS combines results of several threading-based structure prediction algorithms and returns several models with predicted coordinates of alpha carbon atoms. We use only the best full-length model according to ordering given by LOMETS for each sequence. In the second step, the relational representation of peptides' spatial structures is created from the structures returned by LOMETS. In the third step, the feature-construction algorithm RelF is used to construct a set of meaningful structural features. Since RelF had been designed for classification problems, we had to find a way to use it for regression problems like this. We followed a straightforward approach. We enriched RelF with preprocessing in which the training data are split into two sets¹ according to antimicrobial activity - the first set containing peptides with lower-than-median activities, the second set containing peptides with higher-than-median activities. As soon as we have a dataset with at least two classes, RelF can be used for construction of discriminative features. The output of RelF is an attribute-value representation in WEKA format. We also added to these files additional information about dipole moment, proportions of amino acid types and their spatial asymmetries [120]. In the last step, SVM with RBF kernel is trained to give a regression model using the WEKA files generated in *step 3*. Parameters of the regression model are tuned using internal cross-validation. When performing cross-validation, the set of patterns is created separately for each train-test split corresponding to iterations of the cross-validation procedure.

¹ When performing cross-validation, we always split the data taking into account only the training set to avoid information leakage into the independent test set.

	Torrent et al. [124]	Our Regression Model		
	q^2	q^2	q	RMSE
CAMEL	0.65	0.65	0.81	1.23
RANDOM	0.72	0.74	0.87	1.23
BEE	-	0.3	0.61	1.04

Table 20: Experimental results obtained by cross-validation, where q^2 is coefficient of determination, q is correlation coefficient and RMSE is root-mean-square error.

10.4 DATA

We used three datasets to evaluate our antimicrobial-activity prediction method. The first dataset named CAMEL was sourced from Cherkasov et al. [16]. It is composed of 101 antimicrobial peptides with experimentally tested antimicrobial potency. These peptides are rich in leucine and it has been demonstrated that they exhibit high activity against various strains of bacteria. The minimal inhibitory concentrations for these peptides have been averaged over 13 microorganisms. The target variable typically modelled in studies on antimicrobial activity is the *antimicrobial potency* which can be calculated from the average minimal inhibitory concentrations (MIC) according to the following formula from [83]

$$\text{Potency} = \log_2 \frac{1066}{\text{MIC}}.$$

The second dataset named RANDOM was presented by Fjell et al. [32]. It contains 200 peptides with fixed length which are composed of a few amino acids (TRP, ARG and LYS and, more limitedly, LEU, VAL and ILE). Although antimicrobial peptides are actually enriched in these residues, a wide diversity in the amino acid content can be found in natural antimicrobial peptides [13]. The peptides were assayed for antimicrobial activity using a strain of *Pseudomonas aeruginosa*. Fjell et al. did not report absolute MIC values, but only MIC values divided by MIC of Bac2A peptide (to simplify the measurements). Using relative MIC values poses no problem, because it manifests itself only through addition of a constant to the potency values (due to the logarithm).

The last dataset, which we named BEE, was compiled from three different sources: peptides from the venom of the eusocial bee *Halictus sexcinctus* and their analogs [84], peptides from the venom of the eusocial bee *Lasioglossum laticeps* [128] and peptides from the venom of the cleptoparasitic bee *Melecta albifrons* [127]. They contain peptides of length ca. 5 - 15 amino acids. The minimal inhibitory concentrations for these peptides were obtained for *Bacillus subtilis*, *Escherichia coli*, *Staphylococcus aureus*, *Pseudomonas aeruginosa*. We used the average of these values following the methodology of previous works [16, 32]. In some cases, when only lower bounds on MIC were available, we used these values.

These datasets have been also used in Chapter 7 in the evaluation of the algorithm BULL. However, there, we tackled only a restricted classification problem – we focused on predicting which peptides have higher-than-median antimicrobial activity.

10.5 RESULTS

In this section we present experiments performed on real-life data described in Section 10.4. We used a representation of peptides that consisted of literals representing types of the amino acids and literals representing pair-wise distances between the amino acids up to 10 Å. These distances were computed from alpha-carbon coordinates obtained from PDB files computed by LOMETS. We used discretisation of distances with discretisation step 2 Å. We trained support vector machine [15] regression models with RBF kernel selecting optimal C (complexity

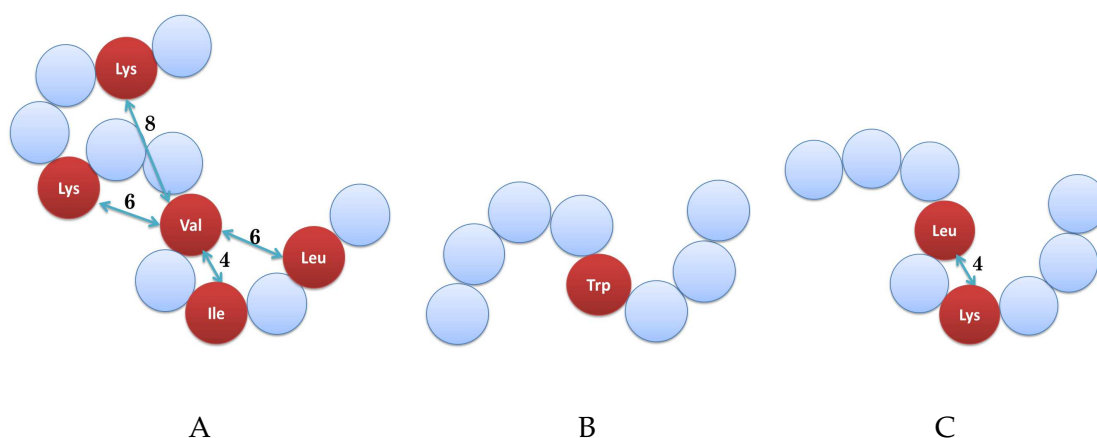


Figure 16: Most informative structural features according to the χ^2 criterion for the dataset of CAMEL (A), RANDOM (B) and BEE (C)(edges not to scale).

constant) and gamma (determines the kernel width parameter) for each fold by internal cross-validation. The estimated results are shown in Table 20.

We performed experiments on three datasets (CAMEL, RANDOM and BEE). We compared the results of our relational learning method for regression with the results reported by Torrent et al. [124] which is a state-of-the-art method. In the work of Torrent et al., only cross-validated coefficients of determination (q^2) were given. Coefficient of determination can be regarded as the proportion of variability in a dataset that is accounted for by the statistical model. In addition, we also report correlation coefficient (q) and root-mean-square error (RMSE) for our regression method. On the dataset CAMEL we achieved the same results as Torrent et al. On the dataset RANDOM we improved upon the results of Torrent et al. in terms of coefficients of determination. Since the dataset BEE is a newly compiled dataset, there are no results to compare our approach with. It is also a harder dataset, than the other two, because it is composed of three different sources. Each of these sources is homogeneous on its own, but the data are heterogeneous when joined. Also the variance of antimicrobial activity is lower in this dataset than in the other two. This explains why the coefficient of determination is so small as compared to the coefficients of determination obtained for the other datasets.

A problem of antimicrobial peptides as antibiotics is that they often have the ability to lyse eukaryotic cells, which is commonly expressed as hemolytic activity or toxicity to red blood cells. Unlike the other methods which use a pre-fixed set of physicochemical features our method is not limited to one particular task. Since the sources from which we compiled the dataset BEE contained also information about the hemolytic activity, we decided to assess the potential of our method also for prediction of hemolytic activity. Because more than half of the reported hemolytic activities were given only by an lower-bound (200 μ M) (i.e. they were not capable to measure the exact value), we decided to transform the problem to a two-class classification problem - the first class corresponding to peptides with activities below the lower-bound, the second class corresponding to peptides with activities higher than the lower-bound. We performed experiments following the same steps as in the prediction of antimicrobial activity, but with a random forest classifier instead of support vector machine classifier for regression. We obtained accuracy 60.83% and AUC (area under ROC curve) 0.725.

In addition, we analysed the structural features used in the regression model in order to get insight into the process by which the antimicrobial peptides kill bacteria. We used the following methodology. First, we discretized the antimicrobial activity attribute, so that we could apply χ^2 criterion for ranking of patterns. Then, for each split of the datasets (CAMEL, RANDOM and BEE) induced by 10-fold cross-validation we selected the three most informative structural features according to the χ^2 criterion. We then selected the pattern which appeared for most of the folds among the three most informative patterns. These patterns are shown in Figure 16 for the three datasets. The pattern selected by this procedure for the dataset CAMEL assumes

presence of five amino acids: ILE, LEU, 2×LYS, VAL with distances between them as depicted in Figure 16. The positively charged Lysines are known to correlate with antimicrobial activity and the presence of leucine can be explained by the fact that the dataset CAMEL contains mostly leucine-rich peptides. Interestingly, the remaining two amino acids - Isoleucine and Valine - and leucine are the only proteinogenic branched-chain amino acids - they each have a carbon chain that branches off from the amino acid's main chain, or backbone. The pattern selected for the dataset RANDOM is very simple. It assumes presence of Tryptophan. Since the patterns count the number of occurrences, it corresponds to proportion of TRP in peptides. This is not surprising, given that the peptides of the dataset RANDOM are composed mostly of TRP and some other amino acids. Finally, the pattern selected for the dataset BEE assumes presence of two amino acids: LEU and LYS in the distance 4Å from each other. Again, the positively charged amino acid - Lysine is known to correlate well with antimicrobial activity. Both leucine and Lysine appeared also in the selected pattern for the dataset CAMEL.

10.6 CONCLUSIONS

We applied relational machine learning techniques to predict antimicrobial activity of peptides. To our best knowledge this study was the first attempt to automatically discover common structural features present in antimicrobial peptides and to use them for prediction of antimicrobial activity. We utilized the feature construction algorithm RelF. There are two main differences between the work presented in this chapter and our earlier work. First, the problem that we tackled in Chapter 9 dealt with classification, whereas here we built a regression model. Second, here only primary structures of peptides are available and therefore we had to rely on structure prediction, whereas we could use spatial structures obtained by X-ray crystallography in our study with DNA-binding proteins. We have shown that our relational learning approach for regression improves on a state-of-the-art approach to antimicrobial activity prediction.

In this chapter, we show that the idea of multivariate polynomial aggregation that we introduced in the relational context (see e.g. Section 4.5) can be used also outside the relational-learning context. Namely, we show that polynomial aggregation features can be used to improve predictive accuracy of a domain-specific method for DNA-binding propensity prediction called the *ball-histogram method* that we introduced in [117].

As we have already mentioned in Chapter 9, improving methods that do not exploit evolutionary information is an important problem. Such methods are valuable mainly due to their ability to predict DNA-binding propensity of proteins for which evolutionary information is not available. One such method based on the algorithm RelF was presented in Chapter 9. Another method not explicitly relying on relational learning, so-called *ball-histogram method*, was described in our work [117]. This method improves upon state-of-the-art in the following way. Rather than relying on a set of features hand-crafted by domain experts, it is based on a systematic, Monte-Carlo-style exploration of the spatial distribution of amino-acids complying to automatically selected properties. For this purpose we employed so-called ball histograms, which are capable of capturing joint probabilities of these specified amino acids occurring in certain distances from each other. A somewhat limiting property of the original ball-histogram method was that it could only work with discrete properties of proteins' regions such as numbers of amino acids of given types. In this chapter we present an approach for dealing with continuous properties of proteins' regions within the ball histogram method. As it turns out, the new method has improved predictive accuracy w.r.t the original ball-histogram method.

This chapter is organized as follows. First, we review the ball-histogram method in Section 11.1. Then we introduce the method for handling continuous properties of protein regions in Section 11.2. We describe experiments on 4 protein datasets in Section 11.3. Section 11.4 concludes the chapter.

11.1 BALL-HISTOGRAM METHOD

In this section we describe the original *ball-histogram method* which has already been applied to prediction of DNA-binding propensity of proteins in [117]. Originally, the motivation for the method was the observation that distributions of certain types of amino acids differed significantly between DNA-binding and non-DNA-binding proteins. This suggested that information about distributions of some amino acids in local regions of proteins could have been used to construct predictive models able to classify proteins as binding or non-binding given their spatial structure. We developed an approach which was able to capture fine differences between the distributions. It consisted of four main parts. First, so-called *templates* were found. In the second step *ball histograms* were constructed for all proteins in a training set. Third, a transformation method was used to convert these histograms to a form usable by standard machine learning algorithms. Finally, a random forest classifier [11] was learned on this transformed dataset and then it was used for classification. In the rest of this section we describe this method in detail.

11.1.1 Ball histograms

A *template* is a list of some Boolean amino acid properties. A property may, for example, refer to the charge of the amino acid (e.g. *Positive*), but it may also directly stipulate the amino acid type (e.g. *arginine*). An example of a template is (Arg, Lys, Polar) or (Positive, Negative, Neutral).

A *bounding sphere* of a protein structure is a sphere with center located in the geometric center of the protein structure and with radius equal to the distance from the center to the farthest amino acid of the protein plus the diameter of the *sampling ball* which is a parameter of the method. We say that an amino acid *falls* within a sampling ball if the alpha-carbon of that amino acid is contained in the sampling ball in the geometric sense.

A ball histogram for a protein P is computed as follows. First, the geometric center C of all amino acids of a given protein P is computed (each amino acid is represented by coordinates of its α -carbon). The radius R_S of the sampling sphere for the protein structure P is then computed as:

$$R_S = \max_{Res \in P} (\text{distance}(Res, C)) + R,$$

where R is a given sampling-ball radius. After that the method collects a pre-defined number of samples containing at least one amino acid from the bounding sphere. For each sampling ball the algorithm counts the number of amino acids in it, which comply with the particular properties contained in a given template and increments a corresponding bin in the histogram. In the end, the histogram is normalized.

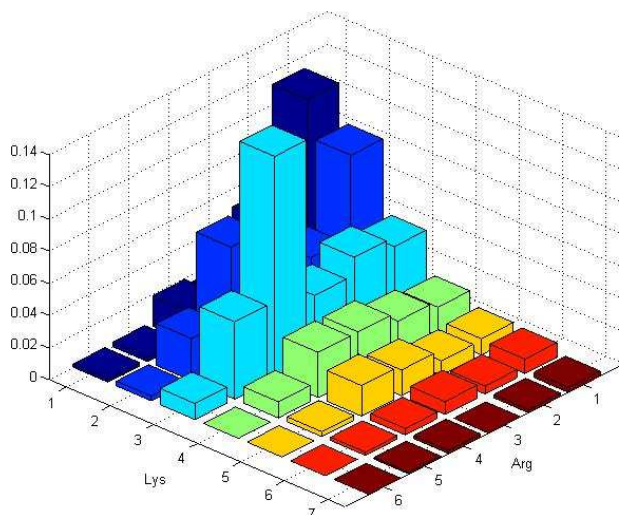


Figure 17: Example ball histogram with template (Arg, Lys) and sampling-ball radius $R = 12 \text{ \AA}$ constructed for protein 1A31 from the dataset PD138.

Example 39. Let us illustrate the process of histogram construction. Consider the template (Arg, Lys) and assume we already have a bounding sphere. The algorithm starts by placing a sampling ball randomly inside the bounding sphere. Assume the first such sampling ball contained the following amino acids: 2 *Arginins* and 1 *leucine* therefore we increment (by 1) the histogram's bin associated with vector $(2, 0)$. Then, in the second sampling ball, we get 1 *histidine* and 1 *Aspartic acid*, so we increment the bin associated with vector $(0, 0)$. We continue in this process until we have gathered a sufficient number of samples. In the end we normalize the histogram. Example of such a histogram is shown in Figure 17.

Ball histograms capture the joint probability that a randomly picked *sampling ball* (see Figure 18) containing at least one amino acid will contain exactly t_1 amino acids complying with property f_1 , t_2 amino acids complying with property f_2 etc. They are invariant to rotation and translation of protein structures which is an important property for classification. Also note that

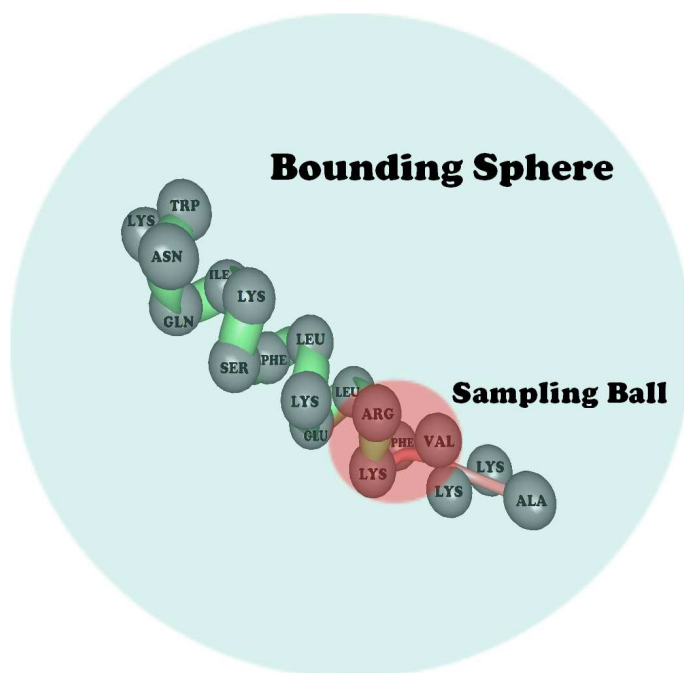


Figure 18: Bounding sphere and sampling ball.

the histograms would not change if we increased the size of the bounding sphere. We make a crisp distinction between the case where an amino acid falls within a sampling ball on one hand, and the case where it falls out of it, on the other hand.

11.1.2 Transformation-based learning

The transformation method for classification is quite straightforward. It looks at all histograms generated from the protein structures in a training set and creates a numerical *attribute* for each vector of property occurrences which is non-zero at least in one of the histograms. After that an attribute vector is created for each training example using the collected attributes. The values of the entries of the attribute-vectors then correspond to heights of the bins in the respective histograms. After this transformation a random forest classifier (or some other classifier able to cope with a large number of attributes) is learned on the attribute-value representation. This classifier is then used for the predictive classification.

11.2 EXTENDING BALL HISTOGRAMS WITH CONTINUOUS VARIABLES

In this section we describe our novel efficient method for incorporation of continuous attributes into the ball-histogram framework. We start by explaining why the existing ball-histogram approach is not suitable for work with continuous attributes. Then we introduce so-called *polynomial aggregation features* and after that we show how they can be used in a ball-histogram-based approach to predictive classification.

11.2.1 Motivation

A drawback of the original ball-histogram method is that it is ill-suited for work with continuous variables. For example, it is possible to model the *distributions* of arginines and Lysines using the ball-histogram method as we have seen in the previous section. There we were able to construct an empirical estimate of the probability $P(\text{Arg} = i, \text{Lys} = j)$ that there are exactly i arginines and j Lysines in a ball randomly sampled from a given protein structure (which can be regarded as the desired *model of the distributions of amino acids* for our predictive purposes). However, if we tried

to model distributions of e.g. *hydropathy* and *volume* of amino acids in a given protein structure in the very same way, we would face serious difficulties stemming from combinatorial explosion of the number of histograms' bins - attributes as the next example indicates.

Example 40. Let us have a protein structure P and a sampling-ball radius R such that any ball can contain at most 6 amino acids. Let us have a template $\tau = [\text{Arg}, \text{Lys}]$. Then there is less than $7^2 = 49$ non-zero bins in the respective histogram for values $[\text{Arg}, \text{Lys}] = [0, 0]$, $[\text{Arg}, \text{Lys}] = [1, 0]$, \dots , $[\text{Arg}, \text{Lys}] = [6, 0]$, \dots , $[\text{Arg}, \text{Lys}] = [0, 6]$. Let us now have a template $\tau_2 = [\text{Hydropathy}, \text{Volume}]$ and note that both hydropathy and volume are properties of amino acids which acquire non-integer values. Now, we can construct a (perhaps pathological and unrealistic) protein structure such that the number of bins will be at least $\binom{20}{6} = 38760$ bins.

This is certainly an impractically high number. Moreover, if we followed the transformation-based approach of the basic ball-histogram method which represents each bin as a real-valued attribute, we would be effectively discarding much of the information about which bins are close to each other (and thus could possibly be considered as one discretized bin).

The above example has illustrated the problems that would arise if we tried to use the original ball-histogram method for continuous properties. A possible approach to cope with these problems would be to use discretization of the values. However, discretization is known to perform poorly when the number of dimensions of the problem increases [17] which may very often be the case with ball histograms. Instead of relying on discretization, we use *multivariate polynomial aggregation* - a strategy that we have recently introduced in the context of statistical relational learning.

11.2.2 Multivariate Polynomial Aggregation Features

Now, we introduce *multivariate polynomial aggregation features*. We start by defining *monomial* and *polynomial features* for sampling balls and then use these definitions to define values of monomial and polynomial features on protein structures.

A *monomial feature* M is a pair $(\tau, (d_1, \dots, d_k))$ where τ is a template with k properties and $d_1, \dots, d_k \in \mathbf{N}$. Degree of M is $\deg(M) = \sum_{i=1}^k d_i$. Given a sampling ball B placed on a protein structure P , we define the *value* of a monomial feature $M = (\tau, (d_1, \dots, d_k))$ as $M(B) = \tau_1^{d_1} \cdot \tau_2^{d_2} \cdot \dots \cdot \tau_k^{d_k}$ where τ_i is the average value of the i -th property of template τ averaged over the amino acids contained in the sampling ball B . We use the convention that $0^0 = 1$. Sometimes, we will use a more convenient notation for monomial features motivated by this definition of *value*:

$$(\tau = (\tau_1, \dots, \tau_k), (d_1, \dots, d_k)) \equiv^{\text{def}} \tau_1^{d_1} \cdot \tau_2^{d_2} \cdot \dots \cdot \tau_k^{d_k}$$

Example 41. Let us have a template

$$\tau = [\text{hydropathy}, \text{volume}],$$

a monomial feature

$$M = \text{hydropathy} \cdot \text{volume}^2$$

and a sampling ball containing two leucines (hydropathy = 3.8, volume = 124) and one arginine (hydropathy = -4.5, volume = 148). Then

$$M(B) = \frac{2 \cdot 3.8 - 4.5}{3} \cdot \left(\frac{2 \cdot 124 + 148}{3} \right)^2 \approx 1.8 \cdot 10^4$$

A *multivariate polynomial feature* is an expression of the form

$$N = \alpha_1 M_1 + \alpha_2 M_2 + \dots + \alpha_k M_k$$

where M_1, \dots, M_k are monomial features and $\alpha_1, \dots, \alpha_k \in \mathbb{R}$ (formally expressed as a pair of two ordered sets - one of monomials and one of the respective coefficients). *Value* of a polynomial feature $N = \alpha_1 M_1 + \dots + \alpha_k M_k$ w.r.t. a sampling ball B placed on a protein structure P is defined as

$$N(B) = \alpha_1 M_1(B) + \alpha_2 M_2(B) + \dots + \alpha_k M_k(B).$$

Degree of a polynomial aggregation feature P is maximum among the degrees of its monomials.

Now, we extend the definitions of values of monomial and polynomial features for protein structures. Given a polynomial aggregation feature N and a sampling-ball radius R , we define the *value* $N(P)$ of a polynomial feature N w.r.t. a protein structure P as:

$$N(P) = \frac{\int_{\hat{P}} N(B) dB}{\int_{\hat{P}} dB} \quad (9)$$

where \hat{P} is the set of all sampling balls with radius R which contain at least one amino acid of the protein structure P . The integral $\int_{\hat{P}} dB$ in the denominator is used as a normalization constant. Intuitively, the integral computes the average value of a polynomial feature N over balls located on a given protein structure. (Although the definition is hopefully intuitively clear, it is not completely specified since we did not define integration over the space of balls. We provide a formal definition in Section 11.5.)

It can be seen quite easily that polynomial aggregation features on protein structures share convenient properties with the discrete ball histograms. They are invariant to rotation and translation of the protein structures which is important for predictive classification tasks. Intuitively, a monomial feature $M = \tau_i$ corresponds to the average value of property τ_i (in sampling balls of a given radius) over a given protein structure. A monomial feature $M = \tau_i^2$ captures the *dispersion* of the values of property τ_i over a given protein structure. Indeed, let us have two proteins A and B and a monomial feature $M = \text{charge}^2$ and let us assume that A and B are composed of the same number of amino acids and that they contain the same number of positively charged amino acids and no negatively charged amino acids. Finally, let us also assume that the positively charged amino acids are distributed more or less uniformly over the protein structure A but are concentrated in a small region of the protein structure B . Then it is not hard to see that for the values $M(A)$ and $M(B)$ it should hold $M(A) \leq M(B)$. Analogically, a monomial feature $M = \tau_i \cdot \tau_j$ corresponds to *agreement* of values of properties τ_i and τ_j over a given protein structure but the *covariance* of these values is better captured by the following expression involving monomial features:

$$M_1(P) - M_2(P) \cdot M_3(P)$$

where $M_1 = \tau_i \cdot \tau_j$, $M_2 = \tau_i$ and $M_3 = \tau_j$. Note that this expression is not a polynomial aggregation feature but only an expression composed of polynomial (monomial) aggregation features. This can be seen when we expand $M_1(P)$, $M_2(P)$ and $M_3(P)$ and obtain

$$M_1(P) - M_2(P)M_3(P) = \frac{\int_{\hat{P}} \tau_i \cdot \tau_j dB}{\int_{\hat{P}} dB} - \frac{\int_{\hat{P}} \tau_i dB}{\int_{\hat{P}} dB} \cdot \frac{\int_{\hat{P}} \tau_j dB}{\int_{\hat{P}} dB}$$

which is not a value of a polynomial aggregation feature. However, it can be easily constructed from some polynomial aggregation features.

Values of polynomial aggregation features can be further decomposed into so called *k-values* computed only from balls containing exactly k amino acids. Given a polynomial feature N and a positive integer k , the k -value of N w.r.t. a protein P is given as

$$N(P|k) = \frac{\int_{\hat{P}_k} N(B) dB}{\int_{\hat{P}_k} dB}$$

where \widehat{P}_k is the set of all sampling balls which contain exactly k amino acids. The value of a polynomial feature can then be expressed using k -values as

$$N(P) = \sum_i \beta_i \cdot N(P|i)$$

where $\beta_i = \int_{\widehat{P}_i} dB / \int_{\widehat{P}} dB$.

In summary, polynomial aggregation features can be expressed using combinations of monomial aggregation features and values of monomial aggregation features can, in turn, be computed using simple expressions involving k -values of monomial aggregation features and proportions of balls containing exactly a given number of amino acids. This implies that when using polynomial features for construction of attributes for machine learning, we can rely solely on the k -values and the few proportions and let the machine learning algorithms compute the values of monomial or polynomial aggregation features from these values if needed.

11.2.3 Ball Histograms with Continuous Variables

Polynomial aggregation features can be used for predictive classification in a way completely analogical to discrete ball histograms. Given a template τ , sampling-ball radius R , a maximum degree d_{\max} and a protein structure, we construct all monomials containing the continuous variables from τ and having degree at most d_{\max} . After that we construct the attribute-table. The rows of this table correspond to examples and the columns (attributes) correspond to k -values of the constructed monomial features. There is an attribute for every k -value such that there is at least one protein structure in the dataset such that it contains a set of k amino acids which fit into a ball of radius R .

The integrals used in definitions of values (or k -values) of monomial aggregation features are difficult to evaluate precisely therefore we use a Monte-Carlo-based approach similar to the case of discrete ball histograms. The set of k -values of monomial aggregation features for a protein P is computed as follows. First, a bounding sphere is found for the protein structure (with geometric center located in the geometric center of the protein structure and with radius $R_S = \max_{Res \in P}(\text{distance}(Res, C)) + R$, where R is a specified sampling-ball radius). After that the method collects a pre-defined number of samples containing at least one amino acid from the bounding sphere. For each sampling ball B the algorithm computes k_B -values (where k_B is the number of amino acids contained in B) of all monomial features complying with a given template and with a given maximum degree and stores them. In the end, the collected k -values of sampling balls are averaged to produce *approximate* k -values for the protein structure P .

After the attribute-table is constructed, it can be used to train an attribute-value classifier such as random forest or support vector machine which can be then used for prediction on unseen proteins.

11.3 EXPERIMENTS

In this section, we describe results obtained in experiments with four datasets of proteins. We compare our novel method to the original ball-histogram method and to the method of Szilágyi and Skolnick [120], which are methods relying only on structural information. We also compare our method with the results of Nimrod et al. [94], which is a method which uses not only structural information but also evolutionary-conservation information.

11.3.1 Data

We used two datasets of DNA-binding proteins and two datasets of non-DNA-binding proteins in our experiments:

- PD138 - dataset of 138 DNA-binding protein structures in complex with DNA,

- UD54 - dataset of 54 DNA-binding protein structures in unbound conformation,
- NB110 - dataset of 110 non-DNA-binding protein structures,
- NB843 - dataset of 843 non-DNA-binding protein structures.

These datasets were described in Section 9.2.

11.3.2 Results

We performed predictive experiments with the four combinations of datasets PD138/ NB110, PD138/NB843, UD54/NB110 and UD54/NB843 described in the previous section. We used monomial aggregation features with maximum degree 3 and the following basic chemical properties of amino acids:

- Amino acid charge (under normal conditions)
- Amino acid Van der Waals volume
- Amino acid hydropathy index
- Amino acid isoelectric point (pI)
- Amino acid dissociation constants pK₁ and pK₂

and the following three properties related to DNA-binding derived by Sathyapriya et al. [106]

- Amino acid base-contact propensity
- Amino acid sugar-contact propensity
- Amino acid phosphate-contact propensity

We trained random forest classifiers using only the attributes having non-zero information gain-ratio on training set. When performing cross-validation, this attribute selection was performed separately on the respective training sets induced by cross-validation so that no information could leak from a training set to a testing set. We compared the *continuous ball-histogram method* presented in this chapter with the original discrete ball-histogram method and with the method of Szilágyi and Skolnick [120], which we reimplemented. The estimated accuracies and AUCs are shown in Table 21. The new continuous ball-histogram method performed best in terms of accuracy in all cases and in terms of AUC in all but one case where the discrete ball-histogram method performed best. We also tested the original ball-histogram method with random forest classifiers enriched with attribute-selection but it did not improve performance.

In addition, we performed experiments with the method of Szilágyi and Skolnick where we replaced logistic regression by random forests (the classifier originally used in their paper was logistic regression for which the obtained accuracy is shown in Table 21). We also compared the obtained results with results reported by Szilágyi and Skolnick in [120]. When using random forest classifier with features of Szilágyi and Skolnick, accuracy increased to 0.82 for the dataset PD138/NB110, which is still lower than 0.89 obtained by the continuous ball-histogram method, and remained unchanged for dataset PD138/NB843 and AUC actually decreased for both of the datasets by 0.02. In [120], AUC 0.93 was reported for the dataset PD138/NB110 which is still lower than 0.95 obtained by the continuous ball-histogram method. Szilágyi and Skolnick also reported AUC 0.91 for the dataset UD54/NB110 in [120] which is higher by 0.01 than the result obtained by the continuous ball histograms. However, this value of AUC was obtained on the dataset UD54/NB110 by a logistic regression classifier trained on the dataset PD138/NB110. The reported value is very probably overoptimistic because the proteins from the dataset NB110 were used both in the training set and in the test set.

	Cont. ball histograms		Disc. ball histograms		Szilágyi et al.	
	Acc	AUC	Acc	AUC	Acc	AUC
PD ₁₃₈ /NB ₁₁₀	0.89	0.95	0.87	0.94	0.81	0.92
PD ₁₃₈ /NB ₈₄₃	0.89	0.86	0.88	0.87	0.87	0.84
UD ₅₄ /NB ₁₁₀	0.87	0.90	0.81	0.89	0.82	0.89
UD ₅₄ /NB ₈₄₃	0.95	0.83	0.94	0.81	0.94	0.78

Table 21: Experimental results obtained by cross-validation for the continuous ball-histogram method (C. ball histograms), the original discrete ball-histogram method (D. ball histograms) and the method of Szilágyi and Skolnick (Szilágyi et al.).

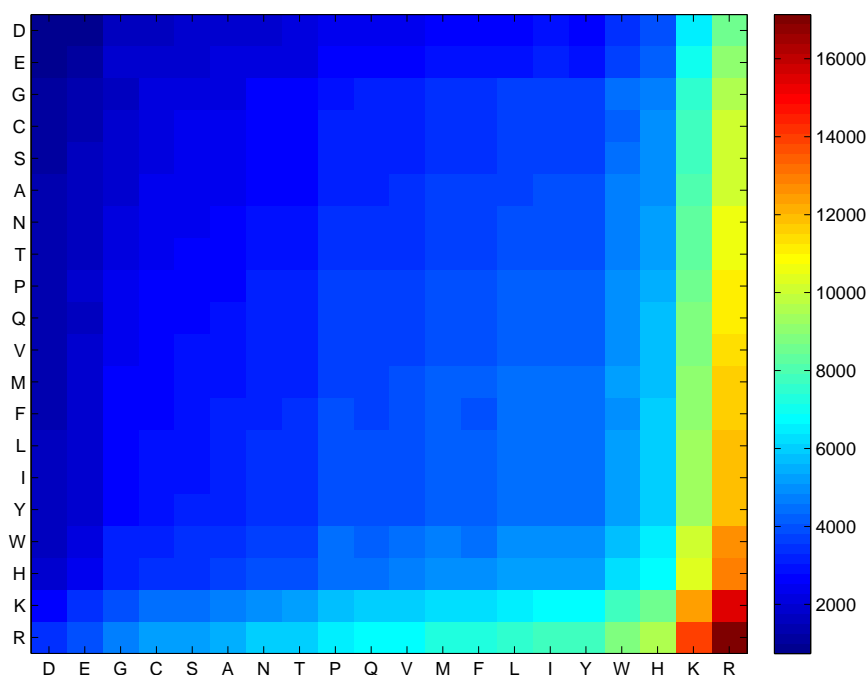


Figure 19: Values of feature volume $\cdot pI^2$ ($k = 3$) for combinations of amino acids in the form $\{AA_1, AA_1, AA_2\}$. The x-axis corresponds to AA_1 and the y-axis corresponds to AA_2 .

In order to see whether the ball-histogram method, which uses only structural information, could come close to the results of methods which exploit also information about evolutionary conservation of regions on protein surfaces, we compared our results with the results of Nimrod et al. [94]. The AUC 0.96 and accuracy 0.90 reported in [94] for the datasets PD₁₃₈ and NB₁₁₀ differs only slightly (by 0.01) from our best results. The AUC 0.90 obtained for the datasets PD₁₃₈ and NB₈₄₃ differs by 0.04 from our best results. These results are encouraging given how important evolutionary information turned out to be according to experiments from [94]. When removing evolutionary information, Nimrod et al.’s misclassification error on the dataset PD₁₃₈/NB₁₁₀ increased by 0.035 which corresponds to lower predictive accuracy than obtained by our method. Moreover, even without the evolutionary information their classifier used significantly more information than our method (e.g. secondary structure information).

In addition to improved accuracy, our method provides us with to-some-extent interpretable features involving distributions of regions with certain chemical properties. We used information-gain attribute selection method to select three most informative attributes on the dataset PD₁₃₈/NB₈₄₃ for further inspection. The selected attributes (i.e. k -values of monomial features) were: volume $\cdot pI^2$ ($k = 3$), $P_p \cdot$ volume \cdot charge ($k = 2$) and $P_p \cdot$ volume $\cdot pI$ ($k = 2$). It is interesting to note that the first (best) monomial did not involve any of the propensities P_p , P_B or P_S and

that only the propensity P_p appeared in the remaining two of the three best monomials. The best feature $\text{volume} \cdot pI^2$ ($k = 3$) defines one value for every combination of three amino acids, which can appear in a sampling ball, and these values are then used to compute aggregated values over protein structures. In Figure 19 we show these values for combinations of amino acids in the form $\{AA_1, AA_1, AA_2\}$. It is interesting to note that the balls corresponding to the highest values of this feature are those containing positively-charged amino acids - arginine and lysine and that, analogically, the balls corresponding to the lowest values are those containing the negatively-charged amino acids - aspartic acid and glutamic acid. This is of course mainly caused by the pI^2 term because pI is the pH at which an amino acid carries no electrical charge. However, it is interesting that the monomial $\text{volume} \cdot pI^2$ is a better predictor of DNA-binding propensity than monomials pI , pI^2 or pI^3 .

11.4 CONCLUSIONS

We have extended our recently introduced *ball histogram method* by incorporation of polynomial aggregation features which are able to capture distributions of continuous properties of proteins' regions. The method achieved higher predictive accuracies than the original ball-histogram method as well as an existing state-of-the-art method. There are interesting future research directions regarding our novel approach. For example, it would be interesting to explore the possibility to use more chemical descriptors of amino acids or protein-regions.

11.5 THEORETICAL DETAILS

In the main text, we used integration over the space of sampling balls without explicitly defining it. We remedy this here. Formally, a sampling ball B is a four-tuple (x, y, z, R) where x, y, z are coordinates of the ball center and R is the ball's radius. Let us have a function $f(x, y, z, R)$ and a set of sampling balls X with fixed radius R and an expression $V = \int_X f(B) dB$. Then V is equal to

$$V = \int_{(x,y,z,R) \in X} f(x, y, z, R) dx dy dz.$$

For example, when we have a monomial feature $M = \text{volume} \cdot \text{charge}$, then

$$f(x, y, z, R) = \frac{\text{avg_volume}_B \cdot \text{avg_charge}_B}{n_B}$$

where n_B , avg_volume_B and avg_charge_B are the number, the average volume and the average charge of amino acids in the sampling ball with center x, y, z and radius R .

Part V

CONCLUSIONS

CONCLUSIONS

In this thesis, we dealt with the problem of making construction of relational features more efficient in the cases when we know or assume that a good set of features can be constructed from features having some special restricted form. We introduced novel algorithms for fast construction of relational features and several other methods and ideas applicable in relational learning in general. We presented novel algorithms of two types: algorithms based on a block-wise strategy exploiting monotonicity of redundancy and reducibility and algorithms based on a new notion of bounded least general generalization.

The main idea underlying the first type of algorithms presented in Part [ii](#) is that redundancy and reducibility can be made monotonous in a certain block-wise feature-construction strategy. The introduced feature-construction algorithms RelF, HiFi and Poly are able to construct exhaustive sets of relatively long non-redundant treelike features. The treelike bias does not compromise the predictive ability. In fact, the new algorithms were able to obtain better predictive accuracies in many domains compared to state-of-the-art relational learning systems. Moreover, they turned out to be useful building blocks of systems for the prediction of DNA-binding propensity of proteins and for the prediction of antimicrobial peptides, which are both important biological problems.

The main idea of the second type of algorithms presented in Part [iii](#) is to generalize existing notions of θ -subsumption, θ -reduction and least general generalization and parametrize them by sets of first-order-logic clauses. Informally, the resulting algorithms then guarantee that the constructed clauses (features) will be at least as *good* as clauses from the set used for parametrization. Importantly, a vast number of local consistency techniques from the field of constraint satisfaction can be used as building blocks of algorithms for various parametrization sets. Two important examples of parametrization sets with the respective local consistency techniques are: clauses of treewidth at most k with the k -consistency algorithm and acyclic clauses with the generalized-arc-consistency algorithm. The experiments that we performed indicate that the new algorithms are able to achieve state-of-the-art accuracy using only small sets of long complex features.

Besides the main contributions which are the feature-construction algorithms, this thesis also introduced polynomial features. We showed that polynomial features are useful in the relational context. In Chapter [11](#), we also showed that they can be useful even outside the relational context. Another smaller contribution presented in this thesis is the introduction of the concept of a safe reduction of a learning example and the development of methods for its utilization as a preprocessing technique. Using the idea of safe reduction, we were able to speed up existing relational learning algorithms by preprocessing their input.

We also described the use of some of the presented algorithms as components of bioinformatics methods for solving the problems of the DNA-binding propensity prediction and the antimicrobial activity prediction of peptides.

Part VI

APPENDIX

The software package *TreeLiker* described in this chapter contains implementations of three feature-construction and propositionalization algorithms: HiFi, RelF (described in Chapter 5), and Poly (described in Chapter 6). All three produce treelike features, use the same mechanism (so-called *templates*) for specifying a particular language bias, and exploit a special, block-wise strategy to construct features. HiFi and RelF generate Boolean-valued or numerical (integer-valued) features. In the latter case, the value is the number of different matching substitutions for the example. Unlike HiFi, RelF assumes that examples are class-labeled and thus is especially suitable for supervised-learning analysis. Using class labels, RelF can filter out *strictly redundant* features. Poly generates numerical (real-valued) features and does not require class labels. Its main distinguishing feature is its focus on domains which contain large amounts of information in the form of numerical data. Poly is based on a technique of multivariate polynomial aggregation.

The algorithms contained in *TreeLiker* have already been used successfully in several bioinformatics studies. RelF was used for prediction of DNA-binding propensity of proteins achieving favourable results as compared to a state-of-the-art method (Chapter 9). It was also used for prediction of antimicrobial activity of peptides for which it also achieved results better than a state-of-the-art method (Chapter 10). Poly was used in preliminary studies to automatize construction of gene-set definitions where it was able to find definitions of gene-sets in terms of gene relations (Chapter 6). These gene-sets turned out to be competitive to gene-sets based on fully-coupled fluxes when used in predictive setting. Poly was also used to construct predictors of DNA-binding propensity that used only information about primary and secondary structure of proteins (Chapter 6).

A.1 IMPLEMENTATION

The algorithms are implemented in Java and all three are based on the same core set of underlying classes. These classes provide support for the syntactical generation of features, for their filtering based on syntactical and redundancy constraints, and for multivariate aggregation. The core classes are compact, comprising of approximately 15 thousand lines of code. The implementation is intended to be easily extendible. For example, it is easy to add new types of multivariate aggregation features. The core code of the algorithms is documented using JAVADOC; there are about 7 thousand lines of comments in the code.

Efficiency of the implemented algorithms was an important objective during the development of *TreeLiker*. The implementation contains substantial parts which are parallelized in order to harness the power of multi-core processors. Furthermore, many sub-results are cached using intelligent mechanism of so-called *soft-references* provided by the Java virtual machine. Soft references allow us to let the virtual machine decide when the cached results should be discarded in order to free the memory.

A.1.1 Representation of Input Data

Learning examples are composed of ground facts which are expressions not involving variables, for example: `hasCharge(arginine)`. Two instances of learning examples are shown below:

DNA-binding aminoacid(a), is(a, histidine), aminoacid(b), is(b, cysteine), distance(a, b, 6.0), distance(b, a, 6.0)

non-DNA-binding aminoacid(a), is(a, tryptophan), aminoacid(b), is(b, tyrosine), distance(a, b, 4.0), distance(b, a, 4.0)

Here, the first *word* on each line denotes class of the example. The rest of the line is then the set of true facts - the description of the example. In this simple case, the first learning example is labelled as a *DNA-binding* protein (that is the class of this example) and the protein has an amino acid Histidine and an amino acid Cysteine which are in distance 6.0 Å from each other. The second learning example is labelled as a *non-DNA-binding* protein and has an amino acid Tryptophan and an amino acid Tyrosine which are in distance 4.0 Å from each other. In a realistic setting, the description of a protein would consist of thousands of facts.

A.1.2 Types of Relational Features

The algorithms contained in TreeLiker (RelF, HiFi and Poly) are intended for construction of relational features. Relational features are conjunctions of literals. For example,

$$F = \text{aminoacid}(A), \text{distance}(A, B, 6.0), \text{is}(B, \text{cysteine})$$

is a feature stipulating the presence of an untyped amino acid and a cysteine in the mutual distance of 6Å. A feature F matches example e if and only if there is a substitution θ to the variables of F such that $F\theta \subseteq e$. So for example, our feature F matches the first learning example in the previous section. This can be also seen as checking whether a conjunctive database query (feature) succeeds for a given relational database (learning example).

There are three settings in which the three feature construction algorithms can work. The first is the existential setting in which we are only interested in whether a given feature matches an example. As a result of matching, the feature thus receives a Boolean value. The second setting is the counting setting. Here, we count how many substitutions θ there are such that $F\theta \subseteq e$ for feature F and example e . The feature then receives an integer value. The counting interpretation showed some significant advantages over the existential interpretation in analysis of DNA-binding proteins [118].

Finally, there is also a third setting based on multivariate relational aggregation, designed especially for representing structures annotated with numerical data. The method is described in detail in Chapters 4 and 6. Briefly, in this setting, a feature may contain several distinguished variables which are used as *extractors* of numerical information. Every substitution θ such that $F\theta \subseteq e$ gives us one sample of the numerical variables which is a vector of real numbers. This vector can be used as input to a multivariate function (which may compute e.g. correlations of the variables). The result is then computed by averaging outputs of the multivariate function over all samples for the given example e .

A.1.3 Language Bias - Templates

Through *templates*, the user constrains the syntax of generated features.¹ Formally, templates are sets of literals. For example, the following expression is a template:

$$\tau_1 = \text{aminoacid}(-a), \text{is}(+a, \#\text{str}), \text{distance}(+a, -b, \#\text{num}), \text{aminoacid}(+b), \text{is}(+b, \#\text{str}).$$

Literals in templates have typed arguments. In template τ_1 above, the types are: a and b . Only same-typed arguments may contain the same variable in a correct feature. For example, the next *feature* complies with the *typing* from template τ_1 :

$$F_1 = \text{aminoacid}(X), \text{distance}(X, Y, 4), \text{is}(Y, \text{histidine})$$

¹ Templates are similar in spirit to *mode-declarations* used in inductive logic programming systems Aleph or Progol [86].

which can be checked easily: variable X appears only in arguments which can be marked by type a and variable Y appears only in arguments which can be marked by type b . On the other hand, the next expression is not a valid feature according to the typing in template τ_1 :

$$F_2 = \text{aminoacid}(X), \text{is}(Y, X)$$

because X appears in arguments marked by two different types: a and str .

Arguments of literals in templates also have *modes*. The most important types of modes are the following (signs, shown in parentheses, are used in templates to denote the particular modes): *input* (+), *output* (-), *constant* (#) and *aggregation* (*).

We start by explaining the two most important types of modes: *input* and *output* modes. Any variable in a valid feature must appear exactly once as an *output* (i.e. in an argument marked by mode -) and at least once as an *input* (i.e. in an argument marked by mode +). For example, F_1 (shown above) complies with this condition w.r.t. the respective templates. On the other hand, for example, F_2 and F_3 and F_4 shown below do not comply with modes specified in template τ_1 :

$$F_3 = \text{aminoacid}(A), \text{distance}(A, B, 6)$$

$$F_4 = \text{distance}(A, B, 6), \text{is}(B, \text{histidine}).$$

The feature F_3 is not valid because variable B appears as an *output* but not as an *input*. The feature F_4 is not valid because variable A appears as an *input* but not as an *output*.

A simple way to understand how templates specify features is to imagine them procedurally as defining a process for constructing valid features. The process can be visualized as follows. We find a literal in the given template which does not contain any input-argument (i.e. none of its arguments is marked by +) and create the first literal of the feature to be constructed from it, e.g.

$$\text{aminoacid}(A) \tag{10}$$

from the template-literal $\text{aminoacid}(-a)$. Then we search for literals in the template which have an input-argument with such type that can be connected to $\text{aminoacid}(A)$. Such a literal is $\text{distance}(+a, -b, \#\text{num})$ or $\text{is}(+a, \#\text{str})$. So, for example, we can create a literal $\text{distance}(A, B, 8)$ according to $\text{distance}(+a, -b, \#\text{num})$ and connect it to $\text{aminoacid}(A)$ which gives us

$$\text{aminoacid}(A), \text{distance}(A, B, 8). \tag{11}$$

Now, we have several options to extend the partially constructed feature 11. One possibility is to connect another literal to the variable A . We can add another literal based on $\text{is}(+a, \#\text{str})$ or $\text{distance}(+a, -b)$, because there may be multiple input-occurrences of one variable, or we can add a literal based on $\text{is}(+b, \#\text{str})$ or $\text{aminoacid}(+b)$ and connect it to variable B . Let us assume that we decided to follow the last option. Then we can get e.g. the following expression:

$$\text{aminoacid}(A), \text{distance}(B, C, 6), \text{is}(B, \text{histidine}) \tag{12}$$

We could continue in this process indefinitely and create larger and larger expressions. However, as we have shown in Chapter 5, there is only a finite number of non-reducible treelike features. Moreover, there are usually even fewer non-reducible and non-redundant features. RelF, HiFi and Poly are able to search through all these possible features exhaustively and efficiently.

Any template-literal can contain at most one input-argument. For example, the next template is not valid

$$\tau_2 = \text{atom}(-a, \#\text{atomType}), \text{bond}(+a, +a)$$

because the literal $\text{bond}(+a, +a)$ has two input arguments.

Mode and type declarations must not contain cycles. There is an additional technical requirement on valid templates. Let us define an auxiliary graph. In this graph, we have one vertex for each type (of arguments) contained in the template. There is an edge from a vertex V to a vertex W if and only if there is a literal which contains the type associated to the vertex V in an input

argument and the type associated to the vertex W in an output argument. This graph must not contain oriented cycles.

This means that the next template

$$\tau_3 = \text{atom}(+a), \text{bond}(+a, -a)$$

is not valid because there is a cycle (loop in this case) from a to a . Similarly, the template

$$\tau_4 = \text{atom}(-a), \text{bond}(+a, -b), \text{bond}(+b, -a)$$

is also not valid because there is a cycle $a - b - a$.

Very often, we need not only variables but also constants. Templates can be used to denote which arguments may contain only constants. For example for the next template

$$\tau_5 = \text{aminoacid}(-a), \text{is}(+a, \#\text{const})$$

one of the possible valid features could be

$$F_5 = \text{aminoacid}(A), \text{is}(A, \text{histidine}).$$

where *his* is a constant. Another example of a template using constants is shown next:

$$\tau_6 = \text{atom}(-a, \#\text{atomType}), \text{bond}(+a, -b), \text{atom}(+b, \#\text{atomType})$$

which specifies features such as:

$$F_6 = \text{atom}(X, \text{carbon}), \text{bond}(X, Y), \text{atom}(Y, \text{carbon}), \text{bond}(X, Z), \text{atom}(Z, \text{hydrogen}).$$

The feature-construction algorithm Poly is able to construct multi-variate polynomial relational features. These are polynomial aggregation features which generalize μ -vectors and σ -matrices from Gaussian logic (see [63]). We need to be able to select which arguments can contain variables that should be used to *extract* the numerical values from the learning examples. We use so-called *aggregation modes* (denoted by $*$) for this. For example the next template:

$$\tau_7 = \text{charge}(-a, *\text{chrg}), \text{bond}(+a, -b), \text{charge}(+b, *\text{chrg})$$

defines features which are able to construct multivariate polynomial features involving charges of atoms in molecules such as:

$$F_7 = \text{charge}(X, \text{CH1}), \text{bond}(X, Y), \text{charge}(Y, \text{CH2}), \text{bond}(X, Z), \text{charge}(Z, \text{CH3})$$

which can in turn be used to construct the polynomial aggregation features such as $\text{AVG}(\text{CH1} \cdot \text{CH2} \cdot \text{CH3})$ or $\text{AVG}(\text{CH1}^2)$.

A.1.4 TreeLiker GUI

The user interacts with TreeLiker either through a graphical interface or through a scripting interface. The former provides only a rather limited access to WEKA's learning algorithms and is meant mainly to assist the user in rapid assessment of the usefulness of the propositionalized representation in the iterations of template tuning. As soon as reasonable settings have been established, the user may employ TreeLiker through the scripting interface within more intricate experimental workflows.

The application consists of six main modules: Input Module, Template Module, Pattern Search Module, Found Patterns Module and Training Module. The Input Module allows the user to select the dataset directories or the specific files that should be used as input data. The user can add as many datasets as desired. The Template Module permits the user to introduce the template specifying the language bias that should be used in the execution of the algorithms.

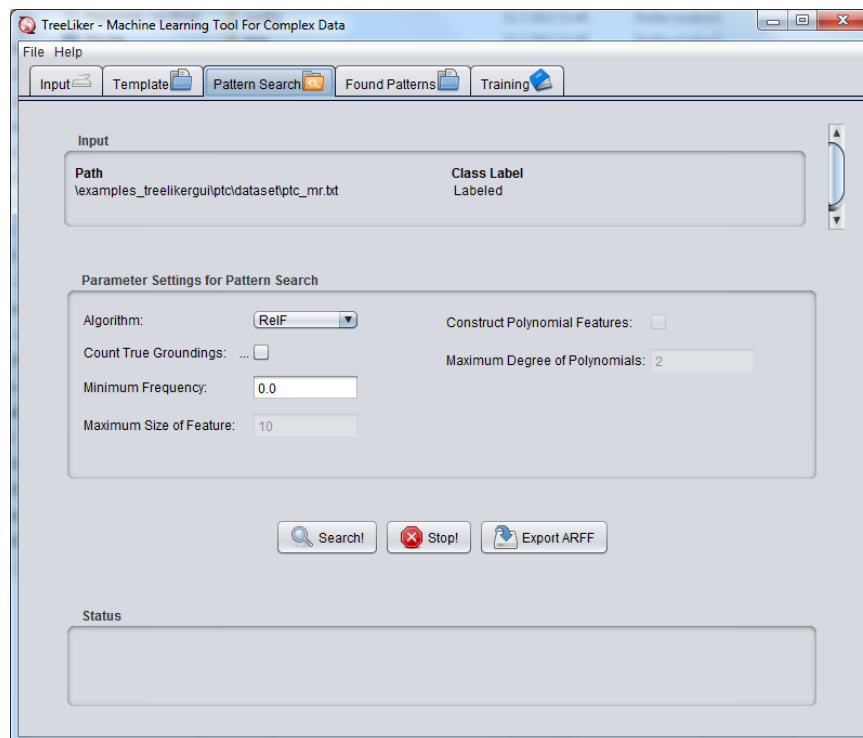


Figure 20: A screenshot of the graphical user interface of TreeLiker.

The Pattern Search Module enables the user to construct relational patterns for the datasets selected in the Input Module. The language bias is taken from the Template Module. The Found Patterns Module uses the results provided by the Pattern Search Module. It shows the structural patterns that were found. The Training Module allows the user to train a classifier based on the patterns generated in the Pattern Search Module. The available classifiers are Zero Rule, SVM with Radial Basis Kernel, J48 Decision Tree, One Rule, Ada-boost, Simple Logistic Regression, Random Forest, L2-Regularized Logistic Regression and Linear SVM.

A screenshot of the graphical user interface is shown in Figure 20.

A.1.5 TreeLiker from Command Line

The command line interface of TreeLiker provides full access to all features of TreeLiker (it provides access to some advanced functionalities which are not accessible from TreeLiker GUI). It allows to set-up more complicated experiments through TreeLiker-batch files.

TreeLiker can be run from command line once we have a TreeLiker-batch file with settings of the experiment. Here, we assume that we already have a TreeLiker-batch file called `experiment.treeliker`. Then, TreeLiker can be run using the following command:

```
java -Xmx1G -jar TreeLiker.jar -batch experiment.treeliker
```

A.1.5.1 TreeLiker-Batch Files

TreeLiker-batch files specify the data to be processed, algorithms with which they should be processed and the detailed settings of the algorithms. The content of a sample TreeLiker-batch file is shown below:

```
set(algorithm, relf) % the algorithm
set(output_type, single) % type of output (single = one file)
set(output, 'trains.arff') % where to save the results
set(examples, 'trains.txt') % the learning examples
```

```
% the template
set(template, [aminoacid(-a), is(+a, #aa_type), aminoacid(+b), is(+b, #aa_type), distance(+a, -b,
#num)])
work(yes) % tells TreeLiker to run the selected algorithm
  % with the selected parameters
```

The first line `set(algorithm, relf)` sets the algorithm to be used by TreeLiker. In this case, it is RelF which works in *existential mode*.

The second line `set(output_type, single)` sets the type of output. There are three types: 1. `single` which constructs one file using all the examples given in the training data, 2. `cv` which creates the given number (10 by default) of pairs of training and testing `.arff` (WEKA) files which can be used to perform cross-validation, 3. `train_test` which creates two files from the given training and testing data.

The third line `set(output, 'trains.arff')` sets the output file in which the constructed relational features and the propositionalized table should be stored. TreeLiker uses `.arff` file format which is used in WEKA.

The fourth line `set(examples, 'trains.txt')` sets path to the training examples which should be used.

The fifth line `set(template, [aminoacid(-a), is(+a, #aa_type), aminoacid(+b), is(+b, #aa_type), distance(+a, -b, #num)])` specifies the template which should be used to constrain the space of possible features.

Finally, the sixth line `work(yes)` tells TreeLiker to run the selected feature construction algorithm.

If the above TreeLiker-batch file is run on the following file of training examples (`trains.txt`):

```
DNA-binding aminoacid(a), is(a, his), aminoacid(b), is(b, cys), aminoacid(c), is(c, arg), distance(a, b, 6.0), distance(b, a, 6.0), distance(a, c, 4.0), distance(c, a, 4.0)
```

```
non-DNA-binding aminoacid(a), is(a, his), aminoacid(b), is(b, cys), aminoacid(c), is(c, arg), distance(a, b, 4.0), distance(b, a, 4.0), distance(a, c, 4.0), distance(c, a, 4.0)
```

```
non-DNA-binding aminoacid(a), is(a, trp), aminoacid(b), is(b, tyr), distance(a, b, 4.0), distance(b, a, 4.0)
```

then it outputs the following `.arff` file:

```
@relation propositionalization
@attribute 'aminoacid(A), distance(A, B, 4.0), aminoacid(B)' {'+'}
@attribute 'aminoacid(A), distance(A, B, 4.0), is(B, cys)' {'+', '-'}
@attribute 'aminoacid(A), distance(A, B, 6.0), aminoacid(B)' {'+', '-'}
@attribute 'aminoacid(A), is(A, arg)' {'+', '-'}
@attribute 'aminoacid(A), is(A, trp)' {'+', '-'}
@attribute 'classification' {'DNA-binding', 'non-DNA-binding'}

@data
'+', '+', '-', '+', '-', 'non-DNA-binding'
'+', '-', '+', '+', '-', 'DNA-binding'
'+', '-', '-', '-', '+', 'non-DNA-binding'
```

In the more realistic settings of our previous experimental evaluations, TreeLiker would construct tens of thousands features for thousands of learning examples. More involved ways of using TreeLiker are described in the user manual.

A.1.6 *Output*

All the three algorithms store their output as .arff files which can be read by WEKA [133]. As exemplified in the previous section, the output file contains both the definitions of the produced features and the attribute-value table consisting of evaluations of each feature on each example.

A.2 AVAILABILITY AND REQUIREMENTS

Project name: TreeLiker

Project home page: <http://sourceforge.net/projects/treeliker>

Operating system: Platform independent

Other requirements: Java 1.6 or higher

License: GNU GPL

Here, we formalize the notion of probability space of learning examples and prove a proposition stated but not proved in the main text of the thesis.

B.1 THE FRAMEWORK

Examples are split into two parts: discrete and continuous. They have both *structure* and *real parameters*. The structure is described by an interpretation, in which the constants r_i represent uninstantiated real parameters. The parameter values are determined by a real vector. Formally, an example is a pair $(H, \vec{\theta})$ where H is an interpretation, $\vec{\theta} \in \Omega_H$, where $\Omega_H \subseteq \mathbb{R}^{|\mathbb{I}(H)|}$ and \mathbb{R} denotes the set of real numbers. The probability space for our learning examples, assuming a given first-order language \mathcal{L} , is the triple (Ω, \mathcal{A}, P) where Ω , \mathcal{A} and P are given as follows.

Let \mathcal{L} be a first-order language with a countable number of predicate symbols, function symbols and constants. Let $\mathcal{H}_{\mathcal{L}}$ be the set of all Herbrand interpretations w.r.t. the language \mathcal{L} with finite length. Note that this set is countable. Since, the set $\mathcal{H}_{\mathcal{L}}$ is countable, we can write $\mathcal{H}_{\mathcal{L}} = \{H_1, H_2, \dots\}$. We set

$$\Omega = \bigcup_{i=1}^{\infty} \{H_i\} \times \mathbb{R}^{|\mathbb{I}(H_i)|}.$$

Next, we set

$$\mathcal{A}_{H_i} = \left\{ \{H_i\} \times M \mid M \in \mathcal{M}(\mathbb{R}^{|\text{Ind}(H_i)|}) \right\}$$

where $H_i \in \mathcal{H}$ is a Herbrand interpretation and $\mathcal{M}(\mathbb{R}^n)$ denotes the σ -algebra of Lebesgue-measurable subsets of \mathbb{R}^n . Then we can define

$$\mathcal{A} = \left\{ \bigcup_{i=1}^{\infty} X_i \mid X_i \in \mathcal{A}_{H_i} \right\}.$$

What we need to show now is that \mathcal{A} is a σ -algebra.

1. $\emptyset \in \mathcal{A}$: Obvious, we can set $X_i = \emptyset$ for all $i \in \mathbb{N}$ (because $\emptyset \in \mathcal{M}(\mathbb{R}^n)$ for any $n \in \mathbb{N}$ and $\{H_i\} \times \emptyset = \emptyset$) and get $\bigcup_{i=1}^{\infty} \emptyset = \emptyset$.
2. If $A \in \mathcal{A}$ then $\Omega \setminus A \in \mathcal{A}$:

$$\Omega \setminus A = \left(\bigcup_{i=1}^{\infty} \{H_i\} \times \mathbb{R}^{|\mathbb{I}(H_i)|} \right) \setminus \left(\bigcup_{i=1}^{\infty} \{H_i\} \times M_i \right)$$

where $M_i \in \mathcal{M}(\mathbb{R}^{|\text{Ind}(H_i)|})$. We can write:

$$\Omega \setminus A = \bigcup_{i=1}^{\infty} \left(\{H_i\} \times (\mathbb{R}^{|\mathbb{I}(H_i)|} \setminus M_i) \right).$$

Now, $(\mathbb{R}^{|\mathbb{I}(H_i)|} \setminus M_i) \in \mathcal{M}(\mathbb{R}^{|\mathbb{I}(H_i)|})$ because $\mathcal{M}(\mathbb{R}^{|\mathbb{I}(H_i)|})$ is a σ -algebra. Therefore also

$$\bigcup_{i=1}^{\infty} \left(\{H_i\} \times (\mathbb{R}^{|\mathbb{I}(H_i)|} \setminus M_i) \right) \in \mathcal{A}.$$

3. If $A_1, A_2, A_3, \dots \in \mathcal{A}$ then $\bigcup_{i=1}^{\infty} A_i \in \mathcal{A}$: We can write

$$A_j = \bigcup_{i=1}^{\infty} X_{i,j}$$

where $X_{i,j} \in \mathcal{A}_{H_i}$. Then the infinite union can be written as:

$$\bigcup_{i=1}^{\infty} A_i = \bigcup_{j=1}^{\infty} \left(\bigcup_{i=1}^{\infty} (\{H_i\} \times M_{i,j}) \right)$$

where $M_{i,j} \in \mathcal{M}(\mathbb{R}^{|\text{Ind}(H_i)|})$. We can continue as follows:

$$\bigcup_{i=1}^{\infty} A_i = \bigcup_{i=1}^{\infty} \{H_i\} \times \left(\bigcup_{j=1}^{\infty} M_{i,j} \right).$$

Since $M_{i,j} \in \mathcal{M}(\mathbb{R}^{|\text{Ind}(H_i)|})$, it must be also the case that $\bigcup_{j=1}^{\infty} M_{i,j} \in \mathcal{M}(\mathbb{R}^{|\text{Ind}(H_i)|})$ because $\mathcal{M}(\mathbb{R}^{|\text{Ind}(H_i)|})$ is a σ -algebra. Therefore we can easily see that $\bigcup_{i=1}^{\infty} A_i \in \mathcal{A}$.

We have shown that \mathcal{A} is a σ -algebra on Ω . Now, we will show how to get a probability measure on this space. Let $P^*(H)$ be a probability function on the countable set of Herbrand interpretations $\mathcal{H}_{\mathcal{L}}$ (some such function certainly exists because the set $\mathcal{H}_{\mathcal{L}}$ is countable). Let $\{f_{H_1}, f_{H_2}, \dots\}$ be a countable set of probability density functions on $\mathbb{R}^{|\text{Ind}(H_i)|}$. Let us define P as follows (this corresponds to the distribution from Eq. 1 which was defined informally in Chapter 4):

$$P \left(\bigcup_{i=1}^{\infty} (\{H_i\} \times M_i) \right) = \sum_{i=1}^{\infty} P^*(H_i) \cdot \int_{M_i} f_{H_i}(\vec{\theta}|H_i) \mathbf{d}\vec{\theta}.$$

Here, $H_i \in \mathcal{H}_{\mathcal{L}}$, $M_i \in \mathcal{M}(\mathbb{R}^{|\text{Ind}(H_i)|})$ and $\mathbf{d}\vec{\theta}$ is Lebesgue measure. Next, we will show that P satisfies Kolmogorov's three axioms of probability.

1. $P(A) \geq 0$ for any $A \in \mathcal{A}$: trivial.
2. $P(\Omega) = 1$:

$$\begin{aligned} P(\Omega) &= P \left(\bigcup_{i=1}^{\infty} (\{H_i\} \times \mathbb{R}^{|\text{Ind}(H_i)|}) \right) = \sum_{i=1}^{\infty} P^*(H_i) \cdot \int_{\mathbb{R}^{|\text{Ind}(H_i)|}} f_{H_i}(\vec{\theta}|H_i) \mathbf{d}\vec{\theta} = \\ &= \sum_{i=1}^{\infty} P^*(H_i) \cdot 1 = 1 \end{aligned}$$

Here, the first equality follows from the fact that f_{H_i} is a probability density function and the second equality follows from the fact that P^* is a probability function.

3. For any countable collection of disjoint sets $A_1, A_2, \dots \in \mathcal{A}$, it holds $P(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$:
We write

$$A_j = \bigcup_{i=1}^{\infty} (\{H_i\} \times M_{i,j})$$

where $M_{i,j}$ is a Lebesgue measurable subset of $\mathbb{R}^{|\text{Ind}(H_i)|}$ then

$$\begin{aligned} \mathbb{P} \left(\bigcup_{j=1}^{\infty} A_j \right) &= \mathbb{P} \left(\bigcup_{j=1}^{\infty} \bigcup_{i=1}^{\infty} (\{H_i\} \times M_{i,j}) \right) = \mathbb{P} \left(\bigcup_{i=1}^{\infty} \left(\{H_i\} \times \bigcup_{j=1}^{\infty} M_{i,j} \right) \right) = \\ &= \sum_{i=1}^{\infty} \mathbb{P}^*(H_i) \cdot \int_{\bigcup_{j=1}^{\infty} M_{i,j}} f_{H_i}(\vec{\theta}|H_i) \, d\vec{\theta} = \sum_{i=1}^{\infty} \mathbb{P}^*(H_i) \cdot \left(\sum_{j=1}^{\infty} \int_{M_{i,j}} f_{H_i}(\vec{\theta}|H_i) \, d\vec{\theta} \right) = \\ &= \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \mathbb{P}^*(H_i) \cdot \int_{M_{i,j}} f_{H_i}(\vec{\theta}|H_i) \, d\vec{\theta} = \sum_{j=1}^{\infty} \sum_{i=1}^{\infty} \mathbb{P}^*(H_i) \cdot \int_{M_{i,j}} f_{H_i}(\vec{\theta}|H_i) \, d\vec{\theta} = \\ &= \sum_{j=1}^{\infty} \mathbb{P} \left(\bigcup_{i=1}^{\infty} (\{H_i\} \times M_{i,j}) \right) = \sum_{j=1}^{\infty} \mathbb{P}(A_j) \end{aligned}$$

Note that the above manipulations are correct because all the involved series converge absolutely.

Thus, we have shown that the triple $(\Omega, \mathcal{A}, \mathbb{P})$ is a probability space.

B.2 PROOFS OF PROPOSITIONS

Next, we prove a proposition from Chapter 4. We will need the following lemma.

Lemma 19. *Let $\mathcal{E} = \{S_1, S_2, \dots\}$ be a countable set of sequences of random variables (estimators) such that:*

1. *any sequence $S_i \in \mathcal{E}$, $S_i = X_1^i, X_2^i, \dots$ converges in mean to the value E^* ,*
2. *for each $S_i, S_j \in \mathcal{E}$, it holds $\mathbf{E}S_i[k] = \mathbf{E}S_j[k]$ where $S_i[k]$ is the k -th element of the sequence S_i .*

Let $\mathcal{E}_m \subseteq \mathcal{E}$ be finite subsets of \mathcal{E} . Then

$$\lim_{m \rightarrow \infty} \Pr \left(\left| E^* - \frac{1}{|\mathcal{E}_m|} \sum_{S_i \in \mathcal{E}_m} S_i[m] \right| > \epsilon \right) = 0$$

where $\epsilon > 0$.

Proof. Let us suppose, for contradiction, that the assumptions of the lemma are satisfied, $\delta > 0$ and that

$$\delta = \lim_{m \rightarrow \infty} \Pr \left(\left| E^* - \frac{1}{|\mathcal{E}_m|} \sum_{S_i \in \mathcal{E}_m} S_i[m] \right| > \epsilon \right) \leq \lim_{m \rightarrow \infty} \Pr \left(\frac{1}{|\mathcal{E}_m|} \sum_{S_i \in \mathcal{E}_m} |E^* - S_i[m]| > \epsilon \right)$$

From this we have

$$\lim_{m \rightarrow \infty} \mathbf{E} \left(\frac{1}{|\mathcal{E}_m|} \sum_{S_i \in \mathcal{E}_m} |E^* - S_i[m]| \right) \geq \delta \cdot \epsilon > 0$$

but

$$\begin{aligned} \lim_{m \rightarrow \infty} \mathbf{E} \left(\frac{1}{|\mathcal{E}_m|} \sum_{S_i \in \mathcal{E}_m} |E^* - S_i[m]| \right) &= \lim_{m \rightarrow \infty} \left(\frac{1}{|\mathcal{E}_m|} \sum_{S_i \in \mathcal{E}_m} \mathbf{E}|E^* - S_i[m]| \right) = \\ &= \lim_{m \rightarrow \infty} \left(\frac{1}{|\mathcal{E}_m|} \cdot |\mathcal{E}_m| \cdot \mathbf{E}|E^* - S_1[m]| \right) = \lim_{m \rightarrow \infty} (\mathbf{E}|E^* - S_1[m]|) = 0 \end{aligned}$$

(where the last equality results from the convergence in mean of the individual random variables) which is a contradiction. The only remaining possibility would be that the limit does not exist but then we can select a subsequence of \mathcal{E}_i which has a non-zero limit and again derive the contradiction as before. \square

Proposition 3. *Let us fix a distribution on examples with probability function $P(H, \Omega_H)$. Let F be a feature and f_F be the density of its sample distribution. Let $q(\theta)$ be a multivariate polynomial. Let X_i be vectors sampled independently from the distribution with density f_F and let $Y_i = \frac{1}{i} \sum_{j=1}^i q(X_j)$. If Y_i converges in mean to \hat{Y} for $i \rightarrow \infty$ then*

$$\tilde{Y}_i = \frac{1}{i} \sum_{j=1}^i Q(e_j)$$

converges in probability to \hat{Y} for $i \rightarrow \infty$ where Q is a polynomial relational feature given by the polynomial $q(\theta)$ and the feature F , and e_i are independently sampled examples.

Proof. First, we will rewrite the formula for \tilde{Y}_i as an average of a (large) number of variables converging in mean to \hat{Y} and after that we will apply Lemma 19. Let us impose a random total ordering on the elements of the sample sets $\mathcal{S}(F, e_i) = \{s_1, \dots, s_{m_i}\}$ so that we could index the elements of these sets. Next, let us have

$$\mathcal{X}_m = \{1, 2, \dots, |\mathcal{S}(F, e_1)|\} \times \{1, 2, \dots, |\mathcal{S}(F, e_2)|\} \times \dots \times \{1, 2, \dots, |\mathcal{S}(F, e_m)|\}$$

Then the formula for \tilde{Y}_i can be rewritten as follows:

$$\tilde{Y}_m = \frac{1}{|\mathcal{X}_m|} \sum_{(j_1, j_2, \dots, j_m) \in \mathcal{X}_m} \frac{1}{m} (q(s_{1, j_1}) + \dots + q(s_{m, j_m}))$$

where $s_{j,k}$ is the k -th element of the sample set $\mathcal{S}(F, e_j)$. Now, each of the summands $Z_m^{(j_1, j_2, \dots, j_m)} = \frac{1}{m} (q(s_{1, j_1}) + \dots + q(s_{m, j_m}))$ is a random variable and any sequence of these summands $Z_m^{(j_1, j_2, \dots, j_m)}$ converges in mean to \hat{Y} for $m \rightarrow \infty$ according to the assumptions of the proposition. Also,

$$\mathbf{E} \left[\frac{1}{m} (q(s_{1, j_1}) + \dots + q(s_{m, j_m})) \right] = \mathbf{E} \left[\frac{1}{m} (q(s_{1, k_1}) + \dots + q(s_{m, k_m})) \right]$$

for all $(j_1, \dots, j_m) \in \mathcal{X}_m$ and $(k_1, \dots, k_m) \in \mathcal{X}_m$. Therefore, we may apply Lemma 19 and infer that \tilde{Y}_m converges in probability to \hat{Y} for $m \rightarrow \infty$. \square

BIBLIOGRAPHY

- [1] S. Ahmad and A. Sarai. Moment-based prediction of dna-binding proteins. *Journal of Molecular Biology*, 341(1):65–71, 2004.
- [2] A. Appice, M. Ceci, S. Rawles, and P. A. Flach. Redundant feature elimination for multi-class problems. In *ICML*, volume 69, 2004.
- [3] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 1 edition, 2009. ISBN 0521424267.
- [4] A. Atserias, A. Bulatov, and V. Dalmau. On the power of k-consistency. In *Proceedings ofICALP-2007*, pages 266–271, 2007.
- [5] G. Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi, and V. Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1999. ISBN 3540654313.
- [6] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of ACM*, 30(3):479–513, July 1983. ISSN 0004-5411.
- [7] C. J. M. Best et al. Molecular alterations in primary prostate cancer after androgen ablation therapy. *Clin Cancer Res*, 11(19 Pt 1):6823–34, 2005.
- [8] H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.
- [9] H. Blockeel, L. Dehaspe, B. Demoen, G. Janssens, J. Ramon, and H. Vandecasteele. Improving the efficiency of inductive logic programming through the use of query packs. *J. Artif. Int. Res.*, 16(1):135–166, 2002. ISSN 1076-9757.
- [10] H. L. Bodlaender and R. H. Mohring. The pathwidth and treewidth of cographs. *SIAM Journal of Discrete Mathematics*, pages 238 – 255, 1993.
- [11] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. ISSN 0885-6125.
- [12] B. Bringmann, A. Zimmermann, L. D. Raedt, and S. Nijssen. Don’t be afraid of simpler patterns. In *PKDD ’06: 10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 55–66. Springer-Verlag, 2006.
- [13] P. Bulet, R. Stöcklin, and L. Menin. Anti-microbial peptides: from invertebrates to vertebrates. *Immunological Reviews*, 198(1), 2004.
- [14] M. E Burczynski et al. Molecular classification of crohns disease and ulcerative colitis patients using transcriptional profiles in peripheral blood mononuclear cells. *J Mol Diagn*, 8(1):51–61, 2006. ISSN 1525-1578.
- [15] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.
- [16] A. Cherkasov and B. Jankovic. Application of ‘inductive’ qsar descriptors for quantification of antibacterial activity of cationic polypeptides. *Molecules*, 9(12):1034–1052, 2004.
- [17] J. Choi, E. Amir, and D. Hill. Lifted inference for relational continuous models. In *Proceedings of the Twenty-Sixth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-10)*, pages 126–134, 2010.

- [18] W. Chu, V. Sindhwani, Z. Ghahramani, and S. Keerthi. Relational learning with gaussian processes. In *Advances in Neural Information Processing Systems 19*, volume 19, pages 289–296, 2007.
- [19] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, March 1990. ISSN 0890-5401.
- [20] Patricia L. M. Dahia et al. A hif1alpha regulatory loop links hypoxia and mitochondrial signals in pheochromocytomas. *PLoS Genet*, 1(1):72–80, 2005.
- [21] J. Davis, E. Burnside, D. Page, I. Dutra, and V. S. Costa. Learning bayesian networks of rules with SAYU. In *Proceedings of the 4th international workshop on Multi-relational mining*. ACM, 2005.
- [22] L. De Raedt. Logical settings for concept-learning. *Artif. Intell.*, 95(1):187–201, 1997.
- [23] L De Raedt. Logical and relational learning. *Springer*, 2008.
- [24] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003. ISBN 1-55860-890-7.
- [25] L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Min. Knowl. Discov.*, 3(1):7–36, 1999. ISSN 1384-5810. doi: <http://dx.doi.org/10.1023/A:1009863704807>.
- [26] B. Dolsak and S. Muggleton. The application of inductive logic programming to finite element mesh design. In *Inductive Logic Programming*, pages 453–472. Academic Press, 1992.
- [27] P. Domingos and P. Singla. Markov logic in infinite domains. In *Probabilistic, Logical and Relational Learning - A Further Synthesis*, 2007.
- [28] P. Domingos, S. Kok, D. Lowd, H. Poon, M. Richardson, and P. Singla. Probabilistic inductive logic programming. chapter Markov logic, pages 92–117. Springer-Verlag, 2008.
- [29] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983.
- [30] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.
- [31] T. Feder and M. Y. Vardi. The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1): 57–104, 1998.
- [32] C. D. Fjell, H. Jenssen, K. Hilpert, W. A. Cheung, N. Pante, R. E. W Hancock, and A. Cherkasov. Identification of novel antibacterial peptides by chemoinformatics and machine learning. *Journal of Medicinal Chemistry*, 52(7):2006–2015, 2009.
- [33] V. Frecer. Qsar analysis of antimicrobial and haemolytic effects of cyclic cationic antimicrobial peptides derived from protegrin-1. *Bioorganic & Medicinal Chemistry*, 14(17):6065 – 6074, 2006.
- [34] W. A. Freije et al. Gene expression profiling of gliomas strongly predicts survival. *Cancer Res*, 64(18):6503–10, 2004.
- [35] E. C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proceedings of the eighth National conference on Artificial intelligence - Volume 1, AAAI'90*, pages 4–9. AAAI Press, 1990.

- [36] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT '95: Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37. Springer-Verlag, 1995. ISBN 3-540-59119-2.
- [37] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *International joint conferences on artificial intelligence*, pages 1300–1309, 1997.
- [38] M. Gao and J. Skolnick. Dbd-hunter: a knowledge-based method for the prediction of dna-protein interactions. *Nucleic Acids Research*, 36(12):3978–92, 2008.
- [39] M. Gao and J. Skolnick. A threading-based method for the prediction of DNA-binding proteins with application to the human genome. *Plos Computational Biology*, 5(11), 2009.
- [40] I. Gashaw et al. Gene signatures of testicular seminoma with emphasis on expression of ets variant gene 4. *Cell Mol Life Sci*, 62(19-20):2359–68, 2005.
- [41] G. J. Gordon. Transcriptional profiling of mesothelioma using microarrays. *Lung Cancer*, 49 Suppl 1:S99–S103, 2005.
- [42] B. Gutmann, M. Jaeger, and L. De Raedt. Extending problog with continuous distributions. In *Inductive Logic Programming*, Lecture Notes in Computer Science, pages 76–91. Springer Berlin / Heidelberg, 2011.
- [43] S. Muggleton H. Lodhi. Is mutagenesis still challenging. In *ILP-05 Late-Breaking Papers*, pages 35–40, 2005.
- [44] R. E. W. Hancock and A. Rozek. Role of membranes in the activities of antimicrobial cationic peptides. *FEMS Microbiology Letters*, 206(2):143–149, 2002.
- [45] P. Hell, J. Nešetřil, and X. Zhu. Duality and polynomial testing of tree homomorphisms. *Transactions of the American Mathematical Society*, 348:1281–1297, 1996.
- [46] C. Helma, R. D. King, S. Kramer, and A. Srinivasan. The predictive toxicology challenge 2000-2001. *Bioinformatics*, 17(1):107–108, 2001.
- [47] J. M. Hilbe. *Logistic Regression Models*. Taylor & Francis, Inc., 2009.
- [48] M. Holec, F. Železný, J. Kléma, and J. Tolar. Integrating multiple-platform expression data through gene set features. In *Proceedings of the 5th International Symposium on Bioinformatics Research and Applications*, ISBRA '09, pages 5–17. Springer-Verlag, 2009.
- [49] T. Horváth and J. Ramon. Efficient frequent connected subgraph mining in graphs of bounded tree-width. *Theoretical Computer Science*, 411(31-33):2784 – 2797, 2010.
- [50] T. Horváth, G. Paass, F. Reichartz, and S. Wrobel. A logic-based approach to relation extraction from texts. In *ILP*, pages 34–48, 2009.
- [51] T. N. Huynh and R. J. Mooney. Max-margin weight learning for markov logic networks. In *ECML/PKDD: European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 564–579, 2009.
- [52] H. Jenssen, T. Lejon, K. Hilpert, C. D. Fjell, A. Cherkasov, and R. E. W. Hancock. Evaluating different descriptors for model design of antimicrobial peptides with enhanced activity toward p. aeruginosa. *Chemical Biology & Drug Design*, 70(2), 2007.
- [53] H. Jenssen, C. D. Fjell, A. Cherkasov, and R. E. W. Hancock. Qsar modeling and computer-aided design of antimicrobial peptides. *Journal of Peptide Science*, 14(1), 2008.
- [54] M. Kanehisa, S. Goto, S. Kawashima, Y. Okuno, and M. Hattori. The kegg resource for deciphering the genome. *Nucleic Acids Research*, 1, 2004.

- [55] K. Kersting. An inductive logic programming approach to statistical relational learning: Thesis. *AI Commun.*, 19:389–390, 2006.
- [56] A. Koopman and A. Siebes. Characteristic relational patterns. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 437–446. ACM, 2009.
- [57] S. Kramer and L. De Raedt. Feature construction with version spaces for biochemical applications. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 258–265. Morgan Kaufmann Publishers Inc., 2001.
- [58] M.-A. Krogel and S. Wrobel. Transformation-based learning using multirelational aggregation. In *ILP '01: Proceedings of the 11th International Conference on Inductive Logic Programming*, pages 142–155, London, UK, 2001. Springer-Verlag. ISBN 3-540-42538-1.
- [59] M.-A. Krogel, S. Rawles, F. Železný, P. A. Flach, N. Lavrač, and S. Wrobel. Comparative evaluation of approaches to propositionalization. In *International Conference on Inductive Logic Programming (ILP 03')*. Springer, 2003.
- [60] M. A. Kuriakose et al. Selection and validation of differentially expressed genes in head and neck cancer. *Cell Mol Life Sci*, 61(11):1372–83, 2004.
- [61] O. Kuželka and F. Železný. Seeing the world through homomorphism: An experimental study on reducibility of examples. In *ILP'10: Inductive Logic Programming*, pages 138–145. 2011.
- [62] O. Kuželka and F. Železný. Block-wise construction of tree-like relational features with monotone reducibility and redundancy. *Machine Learning*, 83:163–192, 2011.
- [63] O. Kuželka, A. Szabóová, M. Holec, and F. Železný. Gaussian logic for predictive classification. In *Proceedings of the European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases, ECML PKDD*, 2011.
- [64] O. Kuželka, A. Szabóová, and F. Železný. Extending the ball-histogram method with continuous distributions and an application to prediction of dna-binding proteins. In *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1–4, 2012.
- [65] O. Kuželka, A. Szabóová, and F. Železný. Relational learning with polynomials. In *IEEE 24th International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 1, pages 1145–1150, 2012.
- [66] O. Kuželka, A. Szabóová, and F. Železný. Bounded least general generalization. In *ILP'12: Inductive Logic Programming*, 2013.
- [67] O. Kuželka, A. Szabóová, and F. Železný. Reducing examples in relational learning with bounded-treewidth hypotheses. In *New Frontiers in Mining Complex Patterns*, pages 17–32, 2013.
- [68] O. Kuželka and F. Železný. A restarted strategy for efficient subsumption testing. *Fundamenta Informaticae*, 89(1):95–109, 2008.
- [69] O. Kuželka and F. Železný. Hifi: Tractable propositionalization through hierarchical feature construction. In Filip Železný and Nada Lavrač, editors, *Late Breaking Papers, the 18th International Conference on Inductive Logic Programming*, 2008.
- [70] O. Kuželka and F. Železný. Fast estimation of first-order clause coverage through randomization and maximum likelihood. In *ICML 2008: 25th International Conference on Machine Learning*, 2008.

- [71] O. Kuželka and F. Železný. Block-wise construction of acyclic relational features with monotone irreducibility and relevancy properties. In *ICML 2009: The 26th International Conference on Machine Learning*, 2009.
- [72] N. Landwehr, M. Hall, and E. Frank. Logistic model trees. *Machine Learning*, 59(1-2): 161–205, 2005.
- [73] N. Landwehr, A. Passerini, L. De Raedt, and P. Frasconi. kFOIL: learning simple relational kernels. In *AAAI'06: Proceedings of the 21st national conference on Artificial intelligence*, pages 389–394. AAAI Press, 2006.
- [74] N. Landwehr, K. Kersting, and L. De Raedt. Integrating naïve bayes and FOIL. *Journal of Machine Learning Research*, 8:481–507, 2007.
- [75] S. Lata, N. Mishra, and G. Raghava. Antibp2: improved version of antibacterial peptide prediction. *BMC Bioinformatics*, 11, 2010.
- [76] N. Lavrač and P. A. Flach. An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic*, 2(4):458–494, 2001. ISSN 1529-3785.
- [77] N. Lavrač, D. Gamberger, and V. Jovanoski. A study of relevance for learning in deductive databases. *Journal of Logic Programming*, 40(2/3):215–249, August/September 1999.
- [78] Michel Liquiere. Arc consistency projection: A new generalization relation for graphs. In *Conceptual Structures: Knowledge Architectures for Smart Applications*, volume 4604, pages 333–346. 2007.
- [79] H. Liu and R. Setiono. Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of IEEE 7th International Conference on Tools with Artificial Intelligence*, pages 338–391, 1995.
- [80] H. Liu, H. Motoda, R. Setiono, and Z. Zhao. Feature selection: An ever evolving frontier in data mining. *Journal of Machine Learning Research - Proceedings Track*, 10:4–13, 2010.
- [81] A. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.
- [82] J. Maloberti and M. Sebag. Fast theta-subsumption with constraint satisfaction algorithms. *Machine Learning*, 55(2):137–174, 2004.
- [83] R. P. Mee, T. R. Auton, and P. J. Morgan. Design of active analogues of a 15-residue peptide using d-optimal design, qsar and a combinatorial search algorithm. *The Journal of Peptide Research*, 49(1), 1997.
- [84] L. Monincová, M. Buděšínský, J. Slaninová, O. Hovorka, J. Cvačka, Z. Voburka, V. Fučík, L. Borovičková, L. Bednárová, J. Straka, and V. Čerovský. Novel antimicrobial peptides from the venom of the eusocial bee halictus sexcinctus (hymenoptera: Halictidae) and their analogs. *Amino Acids*, 39:763–775, 2010.
- [85] J. L. Moreland, A. Gramada, O. V. Buzko, Q. Zhang, and P. E. Bourne. The molecular biology toolkit (mbt): A modular platform for developing molecular visualization applications. *BMC Bioinformatics*, 2005.
- [86] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
- [87] S. Muggleton and C. Feng. Efficient induction of logic programs. In *ALT*, pages 368–381, 1990.

- [88] S. Muggleton, J. Santos, and Alireza Tamaddoni-Nezhad. Prologem: A system based on relative minimal generalisation. In *ILP*, pages 131–148, 2009.
- [89] H Nassif, H Al-Ali, S Khuri, W Keirouz, and D Page. An Inductive Logic Programming approach to validate hexose biochemical knowledge. In *Proceedings of the 19th International Conference on ILP*, pages 149–165, Leuven, Belgium, 2009.
- [90] H. Nassif, H. Al-Ali, S. Khuri, W. Keirouz, and D. Page. An Inductive Logic Programming approach to validate hexose biochemical knowledge. In *Proceedings of the 19th International Conference on ILP*, pages 149–165, Leuven, Belgium, 2009.
- [91] S. Natarajan, T. Khot, K. Kersting, B. Gutmann, and J. Shavlik. Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning*, 83, 2011.
- [92] S.-H. Nienhuys-Cheng and R. de Wolf, editors. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Computer Science*. Springer, 1997. ISBN 3-540-62927-0.
- [93] S. Nijssen and J. N. Kok. The Gaston tool for frequent subgraph mining. *Electronic Notes in Theoretical Computer Science*, 127(1):77–87, 2005.
- [94] G. G. Nimrod, A. Szilágyi, C. Leslie, and N. Ben-Tal. Identification of dna-binding proteins using structural, electrostatic and evolutionary features. *Journal of molecular biology*, 387(4): 1040–53, 2009.
- [95] S. Perkins and J. Theiler. Online feature selection using grafting. In *ICML*, pages 592–599. AAAI Press, 2003.
- [96] B. M. Peters, M. E. Shirtliff, and M. A. Jabra-Rizk. Antimicrobial peptides: Primeval molecules or future drugs? *PLoS Pathogens*, 6, 10 2010.
- [97] G. Plotkin. *A note on inductive generalization*. Edinburgh University Press, 1970. ISBN 80-01-02095-9.
- [98] P. Prosser, K. Stergiou, and T. Walsh. Singleton consistencies. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming, CP '02*, pages 353–368, London, UK, UK, 2000. Springer-Verlag. ISBN 3-540-41053-8.
- [99] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.
- [100] L. De Raedt and K. Kersting. Probabilistic inductive logic programming. In *Probabilistic Inductive Logic Programming - Theory and Applications*, pages 1–27, 2008.
- [101] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.
- [102] B. Rost and C. Sander. Improved prediction of protein secondary structure by use of sequence profiles and neural networks. *Proc. Natl Acad. Sci.*, pages 7558–7562, 1993.
- [103] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.
- [104] J. C. A. Santos and S. Muggleton. Subsumer: A prolog theta-subsumption engine. In *ICLP (Technical Communications)*, pages 172–181, 2010.
- [105] J. C. Almeida Santos, H. Nassif, D. Page, S. H. Muggleton, and M. J. E. Sternberg. Automated identification of protein-ligand interaction features using inductive logic programming: a hexose binding case study. *BMC Bioinformatics*, 13:162, 2012.

- [106] R. Sathyapriya, M. S. Vijayabaskar, and Saraswathi Vishveshwara. Insights into protein-dna interactions through structure network analysis. *PLoS Comput Biol*, 4(9):e1000170, 09 2008.
- [107] T. Scheffer and R. Herbrich. Unbiased assessment of learning algorithms. In *In IJCAI-97*, pages 798–803, 1997.
- [108] C. R. Scherzer et al. Molecular markers of early parkinsons disease based on gene expression in blood. *Proc Natl Acad Sci U S A*, 104(3):955–60, 2007.
- [109] L. Schietgat, F. Costa, J. Ramon, and L. De Raedt. Effective feature construction by maximum common subgraph sampling. *Machine Learning*, 83(2):137–161, 2011.
- [110] M. Sipser. *Introduction to the Theory of Computation, Second Edition*. Course Technology, 2005. ISBN 0534950973.
- [111] A. Srinivasan. The aleph manual, 4th edition. available online: <http://www.comlab.ox.ac.uk/activities/machinelearning/aleph/aleph.htm>, 2007.
- [112] A. Srinivasan and S. H. Muggleton. Mutagenesis: Ilp experiments in a non-determinate biological domain. In *Proceedings of the 4th International Workshop on Inductive Logic Programming, volume 237 of GMD-Studien*, pages 217–232, 1994.
- [113] A. Srinivasan, R. D. King, S. Muggleton, and M. J. E. Sternberg. Carcinogenesis predictions using ILP. In *ILP '97: Proceedings of the 7th International Workshop on Inductive Logic Programming*, pages 273–287. Springer-Verlag, 1997.
- [114] E. W. Stawiski, L. M. Gregoret, and Y. Mandel-Gutfreund. Annotating nucleic acid-binding function based on protein structure. *Journal of Molecular Biology*, 2003.
- [115] S. J. Swamidass, J. Chen, J. Bruand, P. Phung, L. Ralaivola, and P. Baldi. Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. *Bioinformatics*, 21(1):359–368, 2005.
- [116] A. Szabóová, O. Kuželka, F. Železný, and J. Tolar. Prediction of dna-binding proteins from structural features. In *MLSB 2010: 4th International Workshop on Machine Learning in Systems Biology*, pages 71–74, 2010.
- [117] A. Szabóová, O. Kuželka, F. Železný, and J. Tolar. Prediction of dna-binding propensity of proteins by the ball-histogram method using automatic template search. *BMC Bioinformatics*, 13(Suppl 10):S3, 2012.
- [118] A. Szabóová, O. Kuželka, F. Železný, and J. Tolar. Prediction of dna-binding proteins from relational features. *Proteome Science*, 10(1):66, 2012.
- [119] A. Szabóová, O. Kuželka, and F. Železný. Prediction of antimicrobial activity of peptides using relational machine learning. In *IEEE International Conference on Bioinformatics and Biomedicine Workshops (BIBMW 2012)*, 2012.
- [120] A. Szilágyi and J. Skolnick. Efficient prediction of nucleic acid binding function from low-resolution protein structures. *Journal of Molecular Biology*, 358(3):922–933, 2006.
- [121] O. Taboureau, O. H. Olsen, J. D. Nielsen, D. Raventos, P. H. Mygind, and H. Kristensen. Design of novispirin antimicrobial peptides by quantitative structure-activity relationship. *Chemical Biology & Drug Design*, 68(1):48–57, 2006.
- [122] S. Thomas, S. Karnik, R. S. Barai, V. K. Jayaraman, and S. Idicula-Thomas. Camp: a useful resource for research on antimicrobial peptides. *Nucleic Acids Research*, 38:D774–D780, 2010.

- [123] M. Torrent, V. Nogués, and E. Boix. A theoretical approach to spot active regions in antimicrobial proteins. *BMC Bioinformatics*, 10(1), 2009.
- [124] M. Torrent, D. Andreu, V. M. Nogués, and E. Boix. Connecting peptide physicochemical and antimicrobial properties by a rational prediction model. *PLoS ONE*, 6, 02 2011.
- [125] M. Van Leeuwen, J. Vreeken, and A. Siebes. Compression picks item sets that matter. In *PKDD '06: 10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 585–592. Springer-Verlag, 2006.
- [126] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.
- [127] V. Čerovský, O. Hovorka, J. Cvačka, Z. Voburka, L. Bednárová, L. Borovičková, J. Slaninová, and V. Fučík. Melectin: A novel antimicrobial peptide from the venom of the cleptoparasitic bee melecta albifrons. *ChemBioChem*, 9(17):2815–2821, 2008.
- [128] V. Čerovský, M. Buděšínský, O. Hovorka, J. Cvačka, Z. Voburka, J. Slaninová, L. Borovičková, V. Fučík, L. Bednárová, I. Votruba, and J. Straka. Lasioglossins: Three novel antimicrobial peptides from the venom of the eusocial bee lasioglossum laticeps (hymenoptera: Halictidae). *ChemBioChem*, 10(12):2089–2099, 2009.
- [129] C. Vens, A. Van Assche, H. Blockeel, and S. Dzeroski. First order random forests with complex aggregates. In *ILP: Inductive Logic Programming*, pages 323–340, 2004.
- [130] F. Železný. Tractable construction of relational features. In *Znalosti 05, Bratislava*, 2005.
- [131] F. Železný and N. Lavrač. Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, 62(1-2):33–63, 2006.
- [132] J. Wang and P. Domingos. Hybrid markov logic networks. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*. AAAI Press, 2008.
- [133] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.
- [134] M. Wörlein, T. Meinl, I. Fischer, and M. Philippsen. A quantitative comparison of the subgraph miners MoFa, gSpan, FFSM, and Gaston. In *PKDD 2005, 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, LNCS, pages 392–403. Springer, 2005.
- [135] S. Wu and Y. Zhang. Lomets: A local meta-threading-server for protein structure prediction. *Nucleic Acids Research*, 35(10):3375–3382, 2007.
- [136] Z. Xu, K. Kersting, and V. Tresp. Multi-relational learning with gaussian processes. In *IJCAI*, pages 1309–1314, 2009.
- [137] M. Yannakis. Algorithms for acyclic database schemes. In *International Conference on Very Large Data Bases (VLDB '81)*, pages 82–94, 1981.
- [138] M. R. Yeaman and N. Y. Yount. Mechanisms of antimicrobial peptide action and resistance. *Pharmacological Reviews*, 55(1):27–55, 2003.
- [139] M. Žáková, F. Železný, J. Garcia-Sedano, C. M. Tissot, N. Lavrač, P. Křemen, and J. Molina. Relational data mining applied to virtual engineering of product designs. In *ILP06*, volume 4455 of *LNAI*, pages 439–453. Springer, 2007.
- [140] H. Zhao, Y. Yang, and Y. Zhou. Structure-based prediction of dna-binding proteins by structural alignment and a volume-fraction corrected dfire-based energy function. *Bioinformatics*, 26(15):1857–1863, 2010.