KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Probabilistic Logical Models for Large-Scale Hybrid Domains

Irma Ravkic

Supervisor:
Prof. dr. Jesse Davis
dr. ir. Jan Ramon, co-supervisor

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering Science: Computer Science

October 2016

Probabilistic Logical Models for Large-Scale Hybrid Domains

Irma RAVKIC

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering Science: Computer Science

October 2016

Abstract

Statistical relational learning formalisms combine first-order logic with probability theory in order to obtain expressive models that capture both complex relational structure and uncertainty. Despite the significant progress made in this field, several important challenges remain open. First, the expressivity of statistical relational learning comes at the cost of inefficient learning and inference in large-scale problems that contain many objects. Second, while many real-world relational domains are hybrid in that they contain objects that are described by both continuous and discrete properties, little attention has been paid to learning from such data. Third, most formalisms ignore the dynamic nature of real-world problems by considering only the static aspects captured by a single snapshot of time in the dynamic process.

This thesis tries to tackle these shortcomings and makes the following four contributions. First, we propose a graph-sampling based approach that approximately counts the number of pattern occurrences in the data, which enables scaling up parameter learning of statistical relational models. Second, we propose a novel statistical relational learning formalism that models hybrid relational domains. Third, we designed the first structure learning algorithm that is able to learn hybrid relational models. Fourth, we adapted our algorithm to learn temporal dependencies present in the data. We demonstrate the utility of our approaches on several challenging applications, such as planning in a real-world robotics setup, and learning from financial and citation data.

Beknopte samenvatting

Statistisch relationeel leren combineert eerst-orde logica en kansrekening met als doel om expressieve modellen te bekomen die zowel complexe relationele structuur als onzekerheid kunnen voorstellen. Ondanks de gestage vooruitgang in dit onderzoeksdomein zijn er nog belangrijke onopgeloste problemen. Ten eerste heeft de expressiviteit van statistische relationele modellen ook een kost: inferentie- en leeralgoritmes zijn inefficiënt op grootschalige problemen die een groot aantal objecten omvatten. Ten tweede zijn relationele domeinen in de echte wereld vaak hybride: ze bevatten objecten met zowel continue als discrete eigenschappen. Automatisch leren uit zulke data is nog maar weinig onderzocht. Ten derde negeren de meeste formalismen de dynamisch aard van problemen in de echte wereld en beschouwen ze enkel een statische momentopname van het dynamisch verloop.

Dit zijn de beperkingen die we in deze thesis proberen op te lossen. We leveren daartoe de volgende vier bijdragen. Eerst stellen we een aanpak voor die gebaseerd is op het bemonsteren van grafen. Deze aanpak telt benaderend hoe vaak een bepaald patroon voorkomt in de data om zo het leren van parameters in een statistisch relationeel model op te schalen. Vervolgens stellen we een nieuw formalisme voor in statistisch relationeel leren dat dient om hybride relationele domeinen te modelleren. Hierna ontwerpen we het eerste algoritme om de structuur van hybride relationele modellen te leren. Tenslotte passen we ons algoritme aan om te leren welke temporele afhankelijkheden zich voordoen in de data. We tonen het nut van onze aanpak in meerdere uitdagende toepassingen, zoals het plannen in een echte robotica omgeving en het leren uit financiële en bibliografische referentie data.

Acknowledgements

Doing a PhD was a rollercoaster, and I really like rollercoasters. Now at the end of this ride I appreciate all the ups and downs that trained me for my future career and life. The entire experience was new for me: a new country and a second home, new friends, and a new way of living and thinking. With the exception of being away from my mom and my dog Mardi for all this time, I can say that the entire experience was very positive. There are some people whom I would like to thank. Some were taking this ride with me and some were just observing and getting dizzy.

I would first like to thank my promoter Jesse Davis and my co-promoter Jan Ramon. Jesse was always encouraging me in the lows, and praising me in the ups of the PhD studies. I appreciate his detailed feedback, but also casual conversations we would have at the end of our weekly meetings surrounded with Legos. When I would be lost in many small details of the work, he would always find a way to keep the big picture. I would like to thank Jan for being patient when I would need more low-level or mathematical feedback. He would willingly use the whiteboard to illustrate some concepts. This two sometimes complementary approaches contributed a lot to my understanding of problems from two perspectives.

I am grateful for the useful feedback and questions by Professors Luc De Raedt and Marie-Francine Moens during my entire PhD. Special thanks go to my external jury members, Professors Kristian Kersting and Kurt Driessens, whose suggestions for improvements made this thesis more polished and comprehensive. I would like to thank Professor Adhemar Bultheel for chairing both private and public sessions, with the former being more disrupted by my broken voice. Thank you Danny for all the support in the beginning of my studies in Leuven.

This work was boosted with the help of a number of people and grants. Daan gladly provided some emergency mentoring in the beginning. Many thanks go to my friend and co-author Martin who patiently handed over the torch of his work on graph sampling, and to Wannes, Benjamin W. and Benedicte for the nice collaboration. And finally, thanks to my co-author Davide with whom I once spent the entire night in the department playing with the robot arm for the purpose of the paper submission. I would like to thank the Research Fund KU Leuven (OT/11/051) for financially supporting my PhD studies. My special gratitude goes to the Google Anita Borg memorial scholarship for additionally supporting my studies in 2015.

In the beginning of my PhD, on the sunny side of the building, I shared offices with Joris, Jonas, Anton, and later Martin. Thank you for all the fun coffee breaks and table tennis games. Besides being an office mate, Anton has been a very close friend with whom I shared many fun moments. Antoine introduced me to additional cheese and food adventures. Then a new guy came in: Benjamin. It did not take long time before we became good friends and started sharing a passion for Hommelbier and the Cartesian product. Vova, my brother from a Russian mother, came later after I taught him some proper KR. Now, we are both in love with the coco and TWSS jokes. Together with Sebastijan, Nikolina, Jessa, Leander, Eduardo, Christina, Leen, Sergey, Irina and Kurt, we have had some pretty wild parties. In the cooler part of the building I shared my office with Jan (go red devils!) and Tim. Thank you both for all the useful feedback and relaxing conversations. Ventures to spaghetti Bolognese in Alma and coffee breaks would not have been fun without: Siegfried, Angelika, Hendrik, Tias, Vaishak, Vincent, Vince, Francesco, Behrouz, Pedro, Kaja, Anna, Daniele, Thanh, Matthijs, Toon, Samuel. My PhD studies would be unbearable without the volleyball games on Thursday evenings. Wim, Luc, Piet, Hanne, Maarten, Inge, Arturo, Bertram, Bram, Jan, thank you for focusing! There are many people I would like to mention but space ... is the final frontier. Each and every person I met during my years in Leuven left an inspirational trace.

My friends Nedim, Amra, Asmir, Admira, Selena, Dino, Kenan, Izo, Azra, Jasna, Sanja and Maja remained as one of my strongest connections with home. Also big thanks to Renata (bogobog!), Anđelo, Dragana, Ivana, Marjan, Ivan, Josip and Arun who made the adaptation to the new, Belgian, environment easier and fun. Hvala vam puno na podršci! Also thanks to my high school teachers Lejla and Evica who stayed in contact after so many years. I would like to express my gratitude to my adopted family: to Monique for the enormous support and babybels, to Dirk for all the Duvels, and to Sophie, Diego and León for their positiveness.

Special thanks go to my mom, Mirsada, for her patience, inspiration, immeasurable love and support. Mama, hvala ti za sve djeteline s četiri lista. My late father inspired me to be a computer scientist in the first place and he would be happy to read this thesis. In the end I would like to thank my biggest technical and moral supporter throughout this journey and many journeys to come in future: my husband Guy. He has always been there to make me smile and forget about tough times, to offer his help with any work related problem I would encounter, to inspire and motivate me when I doubt my choices, and to give his love every single moment.

Thanks, bedankt and hvala!

Irma Ravkić
Leuven, October 2016

Contents

List of Symbols

f	Feature
λ_G	Labeling function for graph G
$\Leftrightarrow$	Equivalence
L	Literal
X	Logical variable
$\mathcal{A}$	A set of actions
$\mathcal{D}$	Distribution
$\mathcal{P}$	Set of predicates
$value$, $\simeq$	Value of a feature/atom with exactly one grounding substitution
Ω	Sample space
ω	Weight (coefficient)
ϕ	Graph embedding

$\rightarrow$	Implication
Σ	Set of labels
θ	Grounding substitution
D	Data (graph)
$E(G)$	Set of edges in graph G
emb	Set of embeddings
I	Interpretation
$N_G(v)$	Neighbours of vertex v in graph G
$random(\cdot)$	Random variable declaration
$range(X)$	Range of a random variable
S	State
$V(G)$	Set of vertices in graph G
X	Random variable
x	Random variable state
$\mathcal{L}$	A set of logvars
$\mathtt{agg}$	Aggregation function
$\mathtt{A}, \mathtt{G}, \mathtt{H}$	An atom
$\mathtt{C}$	Conjunction
$\mathtt{grsub}$	A set of grounding substitutions
$\mathtt{P}, \mathtt{Q}_{t+1}$	Predicate
$\mathbf{X}$	A set of random variables
$\mathbf{x}$	A state of a set of random variables
G	Graph
$\log$	Logarithm
P	Pattern

P(X)	The probability of a random variable X
p(X)	The probability density of a random variable X
Parents(X)	Parents of a variable X
parents(x)	Assignment to the parents for random variable assignment x
Z	Normalization constant

List of Figures

1 Introduction

"To start, press any key...
Where's the "Any" key?"
Homer Simpson

Real-world applications such as robotics, medicine, molecular biology, and social networks, amongst others, exhibit complex *relational* structure comprising of generic *entities*. Each entity is described by a number of *attributes*, and is *related* to a number of other entities of the same or different type. These relationships are general *patterns* or templates that *match* specific instantiations of objects (e.g., all *humans* are mortal, so is *Socrates*), and can be represented with *first-order logic*. Moreover, the world in which objects interact might be uncertain, which necessitates the tools of *probability*. For example, in *Goodreads*, a social network for book lovers, people are entities that have *discrete* attributes (e.g., favorite book type) and *numeric* attributes (e.g., the average number of books they read per year), and are connected through friendship relations. There also exists uncertainty about the existence and nature of these relations (e.g., there is a 20% chance that people lie about having read a classic or a book that was read by most of their friends). Additionally, the environment where objects reside can exhibit sequential changes or *dynamics*: the number of objects changes, relationships between objects appear and disappear, and the properties of objects evolve as they interact. For example, a user of *Goodreads* can be influenced by the *aggregated* opinion of some friends about a book, causing her to change her rating for the book.

Statistical relational learning (SRL) (Getoor and Taskar 2007; De Raedt, Frasconi, et al. 2008; De Raedt 2008; De Raedt and Kersting 2011), a subfield of machine learning, is concerned with developing formalisms that deal with structured and uncertain domains. The SRL formalisms compactly represent the structure of a domain in the form of regularities or patterns that are quantified by a set of parameters. The parameters of a pattern in SRL are *shared* or *tied* across different instantiations of the pattern, which enables compact representation.

There are several tasks in SRL. The most general and complex task is to *learn* the *structure* that captures the regularities in the domain. The task of *parameter estimation* is used to extract the parameters that quantify the regularities. This two tasks are coupled in that that we want learn the parametrized regularities that best explain the data. Given or learned parametrized models can be used to perform *inference* or answer queries about specific entity instantiations, possibly given some observation or evidence. The approaches in SRL mostly focus on modelling the *static* aspect of the relational data meaning that the data is seen as a snapshot consisting of all the objects and relationships between them at a specific point in time. However, many domains such as robotics have a temporal or *dynamic* aspect. *Planning* approaches for dynamic relational domains devise a sequence of actions based on the state transition model that quantitatively regulates how the attributes and relationships between objects *transition* from one *state* to another.

Despite its progress, many challenges exist in statistical relational learning. One important challenge is to design *scalable* and *efficient* learning algorithms. The inefficiency is related to the choice of the representation and time-consuming structure learning subroutines such as parameter estimation and inference. For example, if one does not care about strictly accurate representations, the structure learning could be made efficient by *decomposing* the process into learning smaller models *independently*. Also, subroutines such as parameter estimation can be performed faster by resorting to *approximations*.

A second challenge lies in the fact that besides exhibiting complex structure, uncertainty and dynamics, real-world domains are also *hybrid*, meaning that they contain both discrete and continuous variables. The usual approach is to discretize continuous variables prior to modelling, learning or inference, which imposes limits on the representation power and leads to a loss of information. This problem raises an interesting question on how to *upgrade existing SRL formalisms to hybrid domains*, and how *to learn and query the hybrid models*.

A third challenge is to *model and learn dynamics in hybrid domains*. The challenge lies in learning *expressive* models that can capture how complex relationships between objects change over time. The learned model can be further used to *perform planning*. Moreover, the algorithms designed to model and plan in dynamic relational domains are usually evaluated on *simulated data*. Thus, the challenge would be to perform structure learning and planning with *real robots acting in dynamic relational environments*.

1.1 Thesis Statement

The goal of this thesis is to push the boundaries of SRL across three dimensions: *scalability*, *expressivity* and *dynamics*. Accordingly, we test the following claims. First, we hypothesize that adapting techniques for approximately counting the number of pattern embeddings in a graph can be adapted to perform scalable and accurate parameter estimation in SRL. Second, the increased *expressivity* of *hybrid* SRL formalisms can provide more accurate and *scalable* structure learning performance than when discretizing the domain prior to learning. And finally, learning *dynamic* relational models in hybrid domains described with relational features and features expressing the arithmetics between low-level attributes, can result in *expressive* models that contribute to good planning performance.

1.2 Contributions

This dissertation addresses the thesis statement by:

1. developing an approximate graph pattern counting approach by extending the algorithm of Fürer and Kasiviswanathan (2008),
2. proposing the hybrid relational dependency networks (HRDNs) as a formalism to model both continuous and discrete variables in relational domains,

3. designing a learner of local models (LLM) to learn the structure of HRDNs,
4. extending LLM to work in dynamic environments.

Several common ideas are interwoven through these topics: extracting and modelling knowledge from rich relational domains, addressing some of the current open questions in SRL, and fusion of different formalisms and applications (e.g., combining graph-based representations with parameter estimation in SRL, using SRL for real-world robotics applications). Next, we present in more detail the contributions for each of the topics addressed in this thesis.

1. Our **first contribution** is to propose a new approach for approximately counting how many times a pattern occurs in a graph. This approach is based on a fully polynomial randomized approximation scheme suggested by Fürer and Kasiviswanathan (2008) for larger unlabeled undirected graphs. As relations of objects and their properties can be represented with graphs, we evaluate this approach on the task of parameter estimation for SRL. This contribution also comprises:
   - Two algorithms for efficiently obtaining an *ordered bipartite decomposition (OBD)* for an input pattern, which is necessary for the theoretical guarantees of the proposed approach to hold.
   - A demonstration of how the count of pattern occurences in a graph can be used to collect the sufficient statistics needed for parameter estimation in SRL.
   - An extensive experimental comparison of the proposed algorithm and two baseline algorithms on the application of parameter estimation in logical Bayesian networks (LBNs) (Fierens, Blockeel, Bruynooghe, et al. 2005), an SRL formalism also used in (Ravkic et al. 2012).
   - An experimental assessment of the approach on *power law* graphs, as opposed to the original theoretical work that only considered *random graphs*.
   - An analysis of the influence of the input pattern and data graph properties on the performance of the proposed and baseline approaches.
   - Two empirical demonstrations showing that the proposed approach: a) converges fast even for large patterns for which exact search fails to

finish in a given time, b) exhibits good performance even for patterns not covered by the theoretical guarantees of the approach.

Using LBNs as an SRL formalism to perform parameter estimation via graph sampling is just an arbitrary choice and other SRL formalisms can be used as well. In essence, the idea behind this approach is simple: the relationships between objects can be represented as graphs and approximately counting graph embeddings in larger datasets can be used to efficiently collect the sufficient statistics.

2. Our **second** main **contribution** are hybrid relational dependency networks (HRDNs), a formalism that upgrades relational dependency networks (RDNs) (Neville and D. Jensen 2007), to hybrid domains. Relational dependency networks are a template language for creating propositional dependency networks (Heckerman et al. 2001) that approximate a joint probability distribution over a set of variables. This contribution includes:
   - The syntax and semantics of hybrid relational dependency networks.
   - A demonstration of local models that can be used for representing the conditional probability distributions that parametrize hybrid dependencies.
   - A discussion of approximate inference for HRDNs.
3. Our **third contribution** is to propose the novel learner of local models (LLM) structure learning algorithm for hybrid relational dependency networks. Since we upgrade relational dependency networks, we inherit their efficient structure learning method that optimizes the conditional probability distribution for each variable independently. This contribution also includes:
   - A discussion on parameter estimation and scoring strategies.
   - An experimental comparison of our approach applied to both hybrid and discretized domains to Markov logic networks (Domingos and Richardson 2004), an SRL approach for modelling discrete data.
   - An empirical demonstration that learning directly in hybrid domains instead of discretizing them prior to learning results in better models.
4. The **fourth** and final main **contribution** presented in this dissertation is the algorithm to learn the structure and the parameters of *dynamic distributional clause (DDC) program* representing a hybrid relational Markov

decision process. In order to realize this, we learn HRDNs (Ravkic et al. 2015) as a proxy to obtain DDCs. Even though this contribution relies on the formalism of HRDNs, in order to meet the needs for handling dynamic hybrid relational models we perform the following upgrades:

- The models are parametrized with relational regression trees. In the initial representation of HRDNs, the models were parametrized with conditional probability tables.
- We introduce relational features that are able to represent arithmetic operations between low-level features. We refer to them as *equational features*.

We then provide the following principal components of this contribution:

- *The DDC Tree Learner*, an algorithm for learning DDCs in hybrid domains.
- A real-world robotics experiment with a robot arm executing different tasks on a number of objects, which represents an interesting application for SRL.
- A real-world evaluation of the learned state transition model by using HYPE (Nitti, Belle, et al. 2015), an existing planning algorithm for hybrid dynamic relational domains.
- A demonstration that the models learned with our approach generalize to unseen cases and that they can grasp interesting relations between objects, which are crucial for accurate prediction and planning.

1.3 Thesis Structure

This work is positioned in the field of statistical relational learning, which is concerned with relational, uncertain and, possibly, dynamic and hybrid domains. In **Chapter 2** we provide necessary background for the components of SRL: *probability distributions*, *modelling propositional data*, *modelling of relational data*, *modelling of statistical relational models*, and finally *modelling dynamic relational data*. For illustrating the attribute-value data format, often used in machine learning, we used the real-world data and models obtained in the kinesiology study we did in:

Irma Ravkic, Benjamin Wittevrongel, Wannes Meert, Jesse Davis, Tim Aderik Gerbrands, Benedicte Vanwanseele "Predicting gait retraining strategies for knee osteoarthritis". Workshop paper at ECML-PKDD conference, September 2015, Porto, Portugal

The next four chapters represent the technical core of this thesis and embody our four main contributions. In **Chapter** 3 we introduce and experimentally evaluate the proposed algorithm for approximately counting pattern embeddings in graphs, based on the theoretical work of Fürer and Kasiwiswanathan, which is then showcased on the task of parameter estimation for SRL. This chapter draws upon the following submission, which is under revision:

Irma Ravkic, Martin Znidarsic, Jan Ramon, Jesse Davis (2016) "Graph sampling for efficient parameter estimation in statistical relational learning". (under the revision at ECML-PKDD Data mining and Knowledge discovery journal)

Chapter 4 introduces hybrid relational dependency networks (HRDNs), our proposed upgrade of relational dependency networks to hybrid domains.

In **Chapter** 5 we introduce the learner of local models (LLM), algorithm for learning the structure of HRDNs. Chapters 4 and 5 are based on the following published work:

Irma Ravkic, Jan Ramon, Jesse Davis (2015) "Learning relational dependency networks in hybrid domains". In: Machine Learning Journal 2015, volume 100, pp. 217 - 254

In **Chapter** 6 we introduce an approach for learning dynamic hybrid relational models and then using the learned models for planning in a real-world robotics application. The work in this chapter is published as:

Davide Nitti, Irma Ravkic*, Jesse Davis, Luc De Raedt (2016) "Learning the structure of dynamic hybrid relational models". In Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI) 2016, Volume 285, pp.1283-1290*

(* *The contributions of this work are equally shared by the authors.*)

Finally, in **Chapter** 7 we summarize our work, provide the main conclusions and give possible directions for future work.

2 Background

"I don't tell you how to tell me what to do, so don't tell me how to do what you tell me to do."
Bender (Futurama)

The previous chapter positioned our work within the field of statistical relational learning and gave a detailed introduction to our contributions. This chapter lays the foundations needed to present these contributions.

SRL combines two components to model complex domains. One component is the *probability distribution* for which we give the foundations in Section 2.1. The second component upgrades the *propositional representations*, which we discuss in Section 2.2, with the notion of related object classes. This upgrade to relational domains can be accomplished by using *relational databases* or *first-order logic* presented in Section 2.3. After introducing these two components of SRL, we combine them together in Section 2.4 by presenting how to model *statistical relational models*. For the purpose of this dissertation we focus on how to model and upgrade *dependency networks*, a *propositional graphical model*, to *relational dependency networks* by using a (restricted) first-order logic. We finalize this chapter by providing the background on *dynamic relational hybrid models* and *planning* in Section 2.5. This involves a brief introduction to *Markov decision processes (MDPs)* and *distributional clauses (DCs)*.

2.1 Probability Distributions

In this section we provide a short introduction to the probability theory notions used throughout this dissertation. A more in depth introduction can be found in Neapolitan (2003).

A *random variable* is a variable that can take on a set of possible values, each of which is associated with a specific probability or density. For example, when throwing a die, each of the six values of the die occurs with the probability of $1/6$. We denote random variables with capital letter (e.g., X). The value or state of a random variable is denoted with a lower case letters (e.g., x). Each random variable has an associated range of values it can take denoted with $range(X)$. The probability that a variable X takes a value $x \in range(X)$ is denoted with $P(X = x)$. We denote sets of random variables by boldface capital letters (e.g., $\mathbf{X}$). The Cartesian product of the ranges of all variables in $\mathbf{X}$ represents *a sample space* denoted as Ω and represents joint assignments to the variables in $\mathbf{X}$. A particular instance of Ω is denoted by boldface lower case letters (e.g., $\mathbf{x}$).

Depending on the range, random variables can be *discrete* or *continuous*. A discrete random variable takes values in a countable set. For example, a die represents a discrete random variable which when tossed can show one of its six possible values. In contrast, a continuous random variable has an infinite number of possible values. For example, room temperature can be a continuous variable taking on real values (e.g., 37.1°).

The probability that a random variable X will take a value x_i and that another random variable Y will take a value y_i is called the *joint* probability of $X = x_i$ and $Y = y_i$ and is denoted as $P(X = x_i, Y = y_i)$. Next we define a *joint probability distribution* over a set of discrete variables, and *joint probability density* over a set of continuous variables.

Definition 1. (Joint probability distribution) *Given a set of discrete random variables $\mathbf{X}$, and the set of all assignments Ω to the variables in $\mathbf{X}$, a joint probability distribution $P(\boldsymbol{x})$ is a function that maps each assignment $\boldsymbol{x} \in \Omega$ to a real number* *such that:*

$$\forall \boldsymbol{x} \in \Omega : 0 \leq P(\boldsymbol{x}) \leq 1$$

and

$$\sum_{\boldsymbol{x} \in \Omega} P(\boldsymbol{x}) = 1$$

Definition 2. (Joint probability density) *Given a set of continuous random variables $\mathbf{X}$, and the set of all the assignments Ω to the variables in $\mathbf{X}$, a joint probability density $p(\boldsymbol{x})$ is a function that maps each assignment $\boldsymbol{x} \in \Omega$ to a real number such that:*

$$\forall \boldsymbol{x} \in \Omega : p(\boldsymbol{x}) \geq 0$$

and

$$\int_{-\infty}^{+\infty} p(\boldsymbol{x}) d\boldsymbol{x} = 1$$

If two random variables X and Y are independent their joint probability distribution is expressed as $P(X, Y) = P(X) \cdot P(Y)$.

Definition 3. (Conditional probability distribution) *Given a joint probability distribution $P(X, Y)$, the conditional probability distribution of X given Y is defined as*

$$P(X|Y) = \frac{P(X, Y)}{P(Y)}$$

and is undefined if $P(Y) = 0$.

A similar definition holds for a *conditional density function*. The difference is that $P(X, Y)$ and $P(Y)$ are replaced with densities denoted $p(X, Y)$ and $p(Y)$, respectively.

Conditional dependencies play a crucial role in probabilistic reasoning systems. First, instead of being interested in the joint probability of events, in real life we are mostly interested in obtaining the probability of an event conditioned on some other event serving as evidence. Second, as we will see in the following section when we introduce probabilistic graphical models, the conditional probabilities are used to factor joint probability distributions leading to efficient representations.

2.2 Modelling Propositional Data

Since its beginning, machine learning approaches have focused on so-called *propositional* or *attribute-value* representations (Mitchell 1997). There are many approaches in machine learning that operate on this format of data, such as decision tree induction (Quinlan 1986), rule induction (Clark and Boswell 1991), and artificial neural networks, amongst others. In this section we illustrate these representations, and we introduce *probabilistic graphical models* (PGMs) (Koller and Friedman 2009) as powerful tools for representing propositional data. In this dissertation we rely on *dependency networks* (DNs) (Heckerman et al. 2001), a PGM that offers an efficient learning strategy.

2.2.1 Single-Table Representation

In many machine learning applications it is natural to assume that data comes in a *single table* with the *attribute-value* format where each row represents an example or an *instance* and columns represent attributes of those instances.

Example 1. *Consider a kinesiology scenario (Ravkic, Wittevrongel, et al. 2015) of tracking a number of healthy patients running with a number of sensors positioned on their body for the purpose of recording the data characterizing their habitual gait. Each patient is then labeled by an expert with a gait retraining strategy that best reduced knee osteoarthritis. The instructions for the patients are to either lean right with the torso (Trunk Lean) or to move the right knee inwards/medial (Medial Thrust). The natural choice for representing this data is the single-table format where each row represents one patient and each column represents attributes describing the patients' habitual gait. A small example of this data is provided in Table 2.1. The task would be to predict the best retraining strategy for a patient given her gait attributes.*

The attribute-value data format is also called *propositional* because each row or example in the single-table format can be described with a fixed-size set of Boolean attributes or *propositions*. Propositions are atomic formulas of propositional logic and can be *true* or *false*. An *interpretation* is a function that assigns *truth values* to the propositions, and each example is one interpretation.

Patient name	*Tibia angle*	*Trunk angle*	*Knee abduction*	...	*Best strategy*
Pete	5.5	3.9	7.0	...	Medial Thrust
Ann	6.3	-1.6	1.5	...	Trunk Lean
Mary	6.7	5.6	-2.4	...	Trunk Lean
		...			

Table 2.1: An illustration of an attribute-value format that in a single table captures a scenario from kinesiology experiment to predict the best gait retraining strategy (*Best strategy*) given a number of attributes of patients' casual gait.

Usually a *closed-world assumption (CWA)* is made meaning that the values of the attributes that are not specified in the data are considered to be false.

Example 2. *The first instance or row in Table 2.1 can be represented as an ordered set of the following propositions {*`Patient name=Pete`*,* `Tibia angle` $= 5.5$*,* `Trunk angle` $= 3.9$*,* `Knee abduction` $= 7.0$*, ... ,* `Best Strategy=Medial Thrust`*}. Under CWA all other propositions for this example, such as* `Tibia angle=`*2.5, are false.*

Decision Trees

One can apply a number of *predictive* or *descriptive* propositional machine learning approaches to the single-table attribute-value format of the data. The predictive approaches aim at learning the model that is good at predicting a *target* attribute on *unseen* data. Descriptive approaches on the other hand do not consider any attribute to be the target but they aim at describing the data or finding general regularities in the data. In this dissertation we use a mixed approach: we aim at extracting general regularities from the data, but underneath we use predictive models to accomplish this.

A widely used predictive model in machine learning are *decision trees* (Quinlan 1986) consisting of internal nodes which represent tests performed on attributes, and leaf-nodes which decide the label of an instance. Decision trees classify instances by sorting them from the root of the tree to a leaf-node reached by following the path established by successful internal node tests. A decision tree that was learned on the full kinesiology data is shown in Figure 2.1. It can be seen that two attributes are selected by the decision tree learning algorithm to be the internal tests. As these are numeric attributes, they are discretized and each branch leads to a specific prediction for the best gait retraining strategy.

There are different tasks in predictive learning depending on whether the target attribute is discrete or continuous. For predicting discrete target attributes one uses *classification*, and for predicting a continuous target attribute one uses *regression*. Consequently, in case the class attribute is continuous the tree is a *regression tree* such that the leaves contain numbers. In other cases we wish to build a decision tree that would be used to estimate the probability distribution over the target attribute given other attributes. The leaves of this kind of regression tree would contain probability distributions which makes them *probability trees*. For example, in the tree in Figure 2.1 the leaves would contain a probability distribution over *Best strategy* values. One branch might state that the probability distribution is $[0.2, 0.8]$ for values $[\textit{Medial Thrust}, \textit{Trunk Lean}]$ given that $-11.5 < \textit{Knee abduction} < -6.0$.

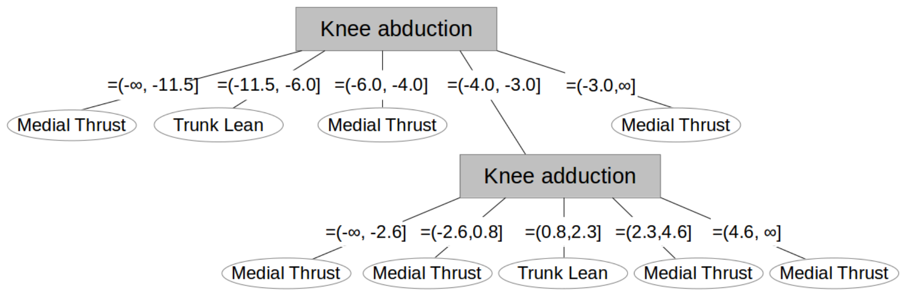

Figure 2.1: A decision tree learned from the dataset in Table 2.1.

Popular descriptive approaches in machine learning are probabilistic graphical models, and we introduce them in more detail in the following section.

2.2.2 Probabilistic Graphical Models

Probabilistic graphical models (PGM) (Koller and Friedman 2009) are a marriage between probability theory and graph theory. They represent a neat framework for compactly representing joint probability distributions with simpler *factors* obtained by exploiting the conditional independencies between random variables. The need for this modularity comes from the fact that the number of parameters needed to encode a joint probability distribution is exponential in the number of random variables. For example, for n binary random variables we would need $2^n - 1$ independent parameters to parametrize the joint probability distribution (the nth is determined from others since probabilities need to sum to one). The most popular PGMs are *Bayesian* (Pearl 1988) and *Markov* (Bishop 2006) networks. We will provide a very short introduction to them, because they are not the main focus of this dissertation. However, we will examine *dependency networks*(DNs) (Heckerman et al. 2001) in more detail, which is a PGM that serves as one of the foundations for this dissertation.

Bayesian Networks

Given a set of random variables, a Bayesian network (Pearl 1988) is a directed acyclic graph $G = (V, E)$ where the nodes V represent a set of random variables $\mathbf{X}$ and the edges represent direct dependencies between the random variables. A Bayesian network graph is parameterized with a set of conditional probability distributions (CPDs) for each $X_i \in \mathbf{X}$ given its parents in the graph, $P(X_i \mid Parents(X_i))$. A probability of a specific value assignments to X_i and $Parents(X_i)$ is denoted $P(X_i = x_i \mid parents(X_i))$.

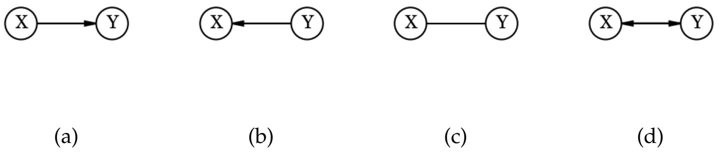

Figure 2.2: Examples of simple dependency graphs that represent a) the Bayesian network encoding $P(X, Y) = P(X)P(Y|X)$, b) the Bayesian network encoding $P(X, Y) = P(Y)P(X|Y)$, c) the undirected graph representing the Markov network, d) the bidirectional dependency graph with $P(X|Y)$ and $P(Y|X)$ conditional distributions.

The structure of a Bayesian network entails some *conditional independencies*: each variable is conditionally independent of all its non-descendants in the graph given the value of all its parents. This property enables a factorization of the joint probability distribution over $\mathbf{X}$ in the following way. Given G together with

CPDs for each random variable, the joint probability for a set of assignments $\mathbf{x}$ for $\mathbf{X}$ is calculated as:

$$P(\mathbf{x}) = \prod_{i=1}^{N} P(X_i = x_i | parents(X_i)) \tag{2.1}$$

Examples of the simplest Bayesian networks are graphs in Figure 2.2a and Figure 2.2b and the joint probability distribution encoded by these graphs are $P(X,Y) = P(X)P(Y|X)$ and $P(X,Y) = P(Y)P(X|Y)$, respectively.

Example 3. *Consider the Bayesian network in Figure 2.2a and the following parameters if we assume that X and Y are Boolean variables: $P(X = True) = 0.2$, $P(Y = True|X = True) = 0.5$, and $P(Y = True|X = False) = 0.2$. Note that we can extract the parameters of other assignments by using the rule of probability theory stating that given a specific condition the probabilities of an event and its negation should sum up to 1. Hence, it holds that $P(X = False) = 1 - P(X = True)$, and $P(Y = False|X = True) = 1 - P(Y = True|X = True)$, etc. The joint probability distribution of this network is shown in Table 2.2 and it can be seen that the joint probabilities over all the assignments (the joint probability distribution) sum to 1. That is, we say that the Bayesian network encodes the proper joint probability distribution over its variables.*

X	Y	$P(X,Y) = P(X) \cdot P(Y\|X)$
$True$	$True$	$0.2 \cdot 0.5 = 0.10$
$True$	$False$	$0.2 \cdot 0.5 = 0.10$
$False$	$True$	$0.8 \cdot 0.2 = 0.16$
$False$	$False$	$0.8 \cdot 0.8 = 0.64$

Table 2.2: The joint probability distribution for the Bayesian network in Figure 2.2a and parameters given in Example 3.

Markov Networks

A Markov network (Bishop 2006) uses an undirected graph $G = (V, E)$ where the nodes in V represent a set of random variables $\mathbf{X}$ and the edges in E correspond to probabilistic interaction or correlation between neighboring random variables. The Markov network is parametrized by a set of *potential functions* $\mathbf{\Phi}$. These potential functions are defined over *cliques* which represent *complete* subgraphs of G. Let $\mathbf{C}$ be a set of cliques of a Markov network G. Each clique $c \in \mathbf{C}$ consists of a set of nodes $\mathbf{X_c}$ and is associated with a clique potential $\phi_c(\mathbf{x_c})$ which is a non-negative function over the possible assignments to $\mathbf{x_c}$. Given a Markov network graph G and a set of potential functions $\mathbf{\Phi}$, the joint probability over random variables of G is expressed as:

$$P(\mathbf{x}) = \frac{1}{Z} \prod_{c \in \mathbf{C}} \phi_c(\mathbf{x_c}) \tag{2.2}$$

where $Z = \sum_X \prod_{c \in \mathbf{C}} \phi_c(x_c)$ is the *normalizing constant*, which ensures that $P(\mathbf{X})$ is a proper joint probability distribution. Note that for the joint probability distribution of Bayesian networks in Equation 2.1 it holds that $Z = 1$ since the acyclicity property ensures the proper joint probability distribution.

Example 4. *A simple example of a Markov network with one clique over X and Y variable is shown in Figure 2.2c. Let the factors of this network be:* $\phi_c(X = True, Y = True) = \phi_c(X = False, Y = False) = 0.6$ *and* $\phi_c(X = True, Y = False) = \phi_c(X = False, Y = True) = 0.4$. *It holds for this example that* $Z = 2$. *To obtain the joint probability, each factor needs to be normalized. For example,* $P(X = True, Y = True) = 0.6/2 = 0.3$. *This indeed ensures that the probability distribution sums to one:* $\sum_x P(x) = 0.6/2 + 0.3/2 + 0.4/2 + 0.4/2 = 1$.

Next we introduce dependency networks in more detail because they will serve as the basis for the work we do in this dissertation.

Dependency Networks

Unlike Bayesian networks and Markov networks, dependency networks (DNs) (Heckerman et al. 2001) *approximate* a joint probability distribution over a set of random variables with a set of conditional probability distributions (CPDs) learned independently. A DN can be represented visually as a directed graph $G = (V, E)$, containing one vertex V_X for each randvar $X \in \mathbf{X}$ and a directed arc from vertex V_X to vertex V_Y iff $X \in Parents(Y)$. Each randvar $X \in \mathbf{X}$ has an associated conditional probability distribution $P(X \mid Parents(X))$, where $Parents(X) \subseteq \mathbf{X} \setminus \{X\}$.

Example 5. *A simple example of a dependency network is shown in Figure 2.2d. Note that the edge between X and Y is bidirectional. There are two CPDs quantifying the dependencies in this example:* $P(X|Y)$ *and* $P(Y|X)$.

A dependency network is said to be *consistent* if there exists a probability distribution P that is consistent with the conditional distributions of the DN. This means that each of the CPDs can be extracted from the joint distribution by applying the rules of probability. If there is no such probability distribution, the DN is called a *general* dependency network and in the remainder of the text we will use the term dependency networks when we think of general DNs. Moreover, the product of CPDs in a general dependency network does not give the joint probability of variables, but their *pseudo-likelihood* (Besag 1974):

$$PL(\mathbf{x}) = \prod_{i=1}^{N} P(X_i = x_i | parents(X_i)). \tag{2.3}$$

This means that dependency networks do not impose that CPDs factor the joint probability distributions and calculating the normalization constant in Equation 2.2 is avoided. The reader interested in literature on compatible conditionals and consistent distributions should read the work of Arnold and Press (1989). Next, we illustrate the consistent and inconsistent DNs in Example 6.

Example 6. *Consider the following CPTs (Lowd 2012) for the dependency network in Figure 2.2d:*

$$\begin{array}{ll} P(X = True|Y = True) = 4/5 & P(X = True|Y = False) = 2/5 \\ P(Y = True|X = True) = 2/3 & P(Y = True|X = False) = 1/4 \end{array}$$

There exists a joint probability distribution

$$\begin{array}{ll} P(X = True, Y = True) = 0.4 & P(X = True, Y = False) = 0.2 \\ P(Y = True, X = True) = 0.1 & P(Y = True, X = False) = 0.3 \end{array}$$

consistent with the CPDs above. The reader can check that for each assignment to values of X and Y it holds that $P(X|Y) = P(X,Y)/P(Y)$ and $P(Y|X) = P(X,Y)/P(X)$.

Now consider the following DN:

$$\begin{array}{ll} P(X = True|Y = True) = 4/5 & P(X = True|Y = False) = 1/5 \\ P(Y = True|X = True) = 1/5 & P(Y = True|X = False) = 4/5 \end{array}$$

The CPD for $P(X|Y)$ encourages X and Y to be equal, while the CPD for $P(Y|X)$ encourages X to be Y different. Hence, this DN is inconsistent and there does not exist a joint distribution from which these CPDs can be extracted.

Parameter Learning. For a given dependency $X_i \mid Parents(X_i)$ the CPD can be any probabilistic regression or classification model and its parameters are estimated by performing *parameter estimation*. Parameter estimation is based on the *maximum likelihood* method which finds the set of parameters that maximize the likelihood, i.e., probability of data given the model. The standard choices for modelling the CPDs are the following:

- **Conditional probability table (CPT)** for a dependency $X_i \mid Parents(X_i)$ provides the probability distributions over values of X_i given the instantiations to the variables in the parent set $Parents(X_i)$. The maximum likelihood estimation of the CPT parameters from data is quite straightforward. For each possible joint instantiation $parents(X_i)$ to the variables in $Parents(X_i)$ we provide probabilities for each value x_i of X_i by counting in the following way:

$$P_{est}(X_i = x_i \mid parents(X_i)) = \frac{N_{x_i,parents(X_i)}}{N_{parents(X_i)}} \tag{2.4}$$

  where $N_{x_i,parents(X_i)}$ represents the number of data instances in which $X_i = x_i$ and $Parents(X_i) = parents(x_i)$. In order to avoid potential problems of extreme probability values of 0 one can use the Laplace correction:

$$P_{est}(X_i = x_i \mid parents(X_i)) = \frac{N_{x_i,parents(X_i)} + 1}{N_{parents(X_i)} + |range(X_i)|} \tag{2.5}$$

  where $range(X_i)$ represents a set of values that variable X_i may attain, and $|range(X_i)|$ represents its cardinality.

- **Logistic regression** gives the probability of a discrete random variable conditioned on a number of discrete or numeric predictors. More generally, if X_i can take on any of the discrete values $\{x_1, \ldots, x_K\}$, then the probability for $X_i = x_1, X_i = x_2, \ldots, X_i = x_{K-1}$ given the instantiation of the parents, $parents(X_i)$, is:

$$P(X_i = x_k \mid parents(X_i)) = \frac{exp(\omega_{k0} + \sum_{y_i \in parents(X_i)} \omega_{ky_i} y_i)}{1 + \sum_{j=1}^{K-1} exp(\omega_{j0} + \sum_{y_i \in parents(X_i)} \omega_{jy_i} y_i)} \tag{2.6}$$

When $X_i = x_K$, it is:

$$P(X_i = x_k \mid parents(X_i)) = \frac{1}{1 + \sum_{j=1}^{K-1} exp(\omega_{j0} + \sum_{y_i \in parents(X_i)} \omega_{jy_i} y_i)} \tag{2.7}$$

To avoid overfitting when training logistic regression the solution is to introduce *regularization* term which penalizes large values of the weights W that parametrize the logistic regression. The regularized approach to learning a logistic regression function is to choose the weights such that the conditional data likelihood is maximized:

$$W \leftarrow \arg\max_{W} \sum_{l} lnP(x_i^l | parents(X_i)^l, W) - \lambda \|W\| \tag{2.8}$$

which adds the penalty proportional to the squared magnitude of W and where x_i^l and $parents(X_i)^l$ represented the values of the respective variables in the lth training example, and λ represents a constant that gives strength to the term.

- **Probability trees** consist of the *internal* nodes representing the binary tests, and *leaves* containing probability distributions for the target variable X_i. The probability in a leaf K is calculated as $\frac{K_{x_i, parents(X_i)}}{K_{parents(X_i)}}$, where $K_{x_i, parents(X_i)}$ represents the number of data instances in the leaf K. *The Laplace correction* can also be applied in the same way as for CPTs.

A problem with using CPTs as the representation for conditional probability distributions is their inefficiency for a larger number of parents: the number of independent parameters needed to encode a CPT grows exponentially with the number of parents. Probability trees offer a sparser representation of CPDs by capturing *context-specific independence* (Boutilier et al. 1996).

Structure Learning. There are three aspects that need to be addressed when performing structure learning. First, one must determine the space of candidate structures. Second one needs to evaluate the "goodness" of different candidate structures. And finally, one needs to define an effective search procedure that finds a good structure. In this thesis we consider a fully observable case and we make a closed world assumption. When learning DNs one optimizes the logarithm of Equation 2.3 or the *pseudo-loglikelihood* (PLL). The PLL of an assignment $\mathbf{x}$ to randvars $\mathbf{X}$ of a DN is calculated as:

$$PLL(\mathbf{x}) = \sum_{i=1}^{N} log\ [P(X_i = x_i | parents(X_i))]. \tag{2.9}$$

Optimizing the PLL has the advantages that it can be decomposed into maximizing the loglikelihood for each variable independently and calculating it does not require computing the partition function (that is, summing over all possible configurations of the randvars). This has a significant impact on the computational efficiency of the search algorithm. Note that this learning procedure is not possible for Bayesian networks since it could lead to a cyclic structure which is not allowed. In other words, learning the structure of a DN from data requires determining $Parents(X)$ for each $X \in \mathbf{X}$ (i.e., the dependency structure) and the parameters of the CPD for X that maximize PLL. For example, one can consider algorithms based on greedy hill-climbing search driven by the score metric: a current candidate structure is maintained and iteratively improved until the score does not increase (Getoor, Friedman, et al. 2001).

One issue with structure learning of probabilistic graphical models is *overfitting*: if we perform likelihood maximization, the most complex structures will always be favored. There are ways to avoid this problem. First, using the Bayesian approach (Heckerman 1999) that uses a prior distribution over the parameters can smooth the irregularities in the training data. Second, one can add hard constraints by selecting a less expressive hypothesis class (e.g., limit the number of parents in the parent set). Third, a simpler model can be preferred which corresponds to the principle of *Occam's razor*: introduce a *penalty* term that penalizes complex structures. One such adaptation of the scoring function is the *minimum description length (MDL)* principle (Schwarz 1978), based on ideas from information theory, which leads to the best compression of the data. Finally, the objective function can be augmented with a *regularization* term.

Inference. Learning each CPD independently could result in an inconsistent model. That is, there may be no joint probability distribution such that it is possible to apply the rules of probability to the joint distribution in order to derive each learned CPD. Regardless of whether a DN is consistent, applying an *ordered* Gibbs sampler to the DN's CPDs results in a unique distribution, given that each variable in the DN is discrete and each CPD in the DN is positive (Heckerman et al. 2001). If the DN is consistent, then its conditional distributions must be consistent with a Markov network, and Gibbs sampling converges to the same distribution as that of the Markov network. In case of an inconsistent DN, the stationary distribution may depend on the order in which variables are sampled, and the joint distribution determined by the Gibbs samples may be inconsistent with the CPDs.

Ordered Gibbs sampling randomly selects the initial value for each random variable, and then in each Gibbs sweep iterates over the variables in a fixed order and resamples the value of each X_i from its local distribution $P(X_i|Parents(X_i))$. If the DN is consistent, it generates the joint probability distribution. If the DN is inconsistent, this procedure is called an *ordered pseudo-Gibbs* sampler (Heckerman et al. 2001).

2.3 Modelling Relational Data

A disadvantage of the attribute-value representation is that many real-world problems, such as ones in chemoinformatics (King et al. 2001), are relational and it is often hard, too lossy and incomprehensible to force the data about a particular example into a single row. This approach is usable for simple domains such as presented in Table 2.1. However, a more complex problem (e.g., the patients are related, a single patient has multiple observation sessions under different conditions and labs) cannot be represented by a single table. In this case one would need to use representations adequate for modelling *relational* domains.

2.3.1 Relational Data and Databases

A relational database is a database that consists of multiple related tables. Each table represents a relationship between classes of objects or their attributes. Each entry in a table is characterized with a unique identifier called the *primary key*. The connection between tables is accomplished by means of *foreign keys*: one or more columns in a table that are primary keys in some other table. The relationships between tables can be of several types. If one row in a table corresponds to exactly one row in another table, we call it a *one-to-one* relationship. In case one row in a table corresponds to multiple rows in another table we call it a *one-to-many* relationship. Finally, the connection when multiple rows in one table correspond to multiple rows in another table is called a *many-to-many* relationship.

We will now introduce a toy example of a slightly modified *university domain* introduced by Getoor, Friedman, et al. (2001) that will be used as an example throughout the thesis.

Example 7. *Consider a university consisting of students, professors and courses. Students have an intelligence and can be friends with other students. A student can take a number of courses and express her satisfaction with each course she took and for which she got a grade. A course has a specific difficulty and a designated number of hours a student should spend on preparing it. Each course is taught by a professor who has a specific ability. A relational schema and a small number of examples is presented in Table 2.3. The underlined columns represent primary keys, and bold column names denote foreign keys. Note that the foreign key in the RECORD table is a combination of the student and the course names. We can notice a couple of relationships between the tables. These relationships are accomplished through foreign keys. For example, the student column is the primary key in the STUDENT table, but a foreign key in the FRIENDS and the RECORD tables. TEACHES and RECORD are an example of a many-to-many connection: each professor can teach multiple courses, and a course can be taught by multiple professors; also each student can take multiple courses, and a course can be taken by multiple students.*

Relational data can be processed in three different ways.

1. The first way is to *flatten* or *propositionalize* (Kramer et al. 2000) the relational data prior to its usage, and thus obtain the attribute-value format we illustrated in Section 2.2.1. Advantages of this approach are its simplicity and the fact that there exists an abundance of propositional learning approaches. However, one disadvantage of this approach is that it produces propositional data with many attributes. This occurs due to the complicated relationships that might exist between objects, such as one-to-many and many-to-many relationships.

2. Instead of propositionalizing data prior to learning, features can be constructed while learning and constructed in such a way that they represent attributes of general object classes and relations between them.

STUDENT

student	intelligence
bob	105.0
ann	115.0
mary	107.0
jerry	112.0

PROFESSOR

professor	ability
john	high
susan	medium

TEACHES

professor	course
john	biology
john	chemistry
susan	math

COURSE

course	difficulty	numHours
biology	medium	90.0
math	high	140.0
chemistry	high	100.0

FRIENDS

student	student
bob	ann
bob	mary
ann	jerry

RECORD

recordID	student	course	grade	satisfaction
rec1	bob	biology	low	medium
rec2	bob	math	medium	high
rec3	ann	biology	high	high
rec4	ann	math	high	medium

Table 2.3: A simplified example of a relational database inspired by the university example described in Example 7.

A tool that would enable this kind of modelling is *first-order* or *predicate* logic. A natural way to solve these tasks is the by using the inductive logic programming (ILP) approaches (Muggleton and De Raedt 1994). ILP is a subfield of machine learning that addresses relational learning. In principle, ILP allows induction over relational structures which can make the learning intractable. Thus, different heuristics are used in ILP methods (Muggleton and De Raedt 1994) to control the search, and learn the concept efficiently. While there are a number of approaches for learning ILP programs, unless restrictions on the hypothesis space are made, the learning might be intractable, inflexible and inefficient in large scale domains.

3. Another option is the combination of relational learning with propositional approaches (Roth and Yih 2001; Davis, Burnside, Castro Dutra, et al. 2005; Landwehr, Kersting, and De Raedt 2005; Landwehr, Kersting, and Raedt 2007), which is the strategy we adopt in this thesis as well. The

general idea behind this approach is to encode relational structures as propositional representations in order to support flexible and efficient learning and evaluation, but to express the learned concepts in relational representations. While in ILP relational features are generated as a part of the structure search, in this method, the relational features are calculated up front.

2.3.2 Representations Based on First-Order Logic

Unlike propositional logic where we only have propositions that can be true or false, in first-order logic the notion of an object class is introduced, which together with quantification (existential and universal) represents an adequate tool (Lloyd 2012) to compactly represent complex relational domains.

In this dissertation we use a variation of datalog, a type of restricted first-order logic. The alphabet of our language consists of three types of symbols: constants, logical variables, and predicates. A *constant* represents a specific object and is denoted with a lower-case letter (e.g., pete). A *logical variable* (*logvar*) X is a variable ranging over the objects in the domain. Logical variables may be *typed* in which case they represent placeholders for a specific subset of objects in the domain. *Predicate* symbols $\mathtt{P}/n$, where $n \geq 0$ is the arity of the predicate, represent properties of objects or relations among objects. Each predicate P has a finite range, denoted $range(\mathtt{P})$. In traditional logic, the range of a predicate is $\{false, true\}$. However, in this thesis we allow a predicate to have a categorical or numeric range. For example, the range of a student's intelligence could be $\{\mathtt{low}, \mathtt{med}, \mathtt{high}\}$ or a range that is the interval of $[0, 180]$. An *atom* is of the form $\mathtt{P}(\mathtt{t_1}, \ldots, \mathtt{t_n})$ where P/n is a predicate and each $\mathtt{t_i}$ is a constant or a logvar. In a *typed* language every argument position of an atom has a type. For example, in atom $grade(S, C)$ we can constraint that logvar S ranges only over *students* and C only over the *courses*. The range of an atom is the range of its predicate. A *literal* is an atom or its negation. An atom is *ground* if all its arguments are constants. A *substitution*, denoted $\{\mathtt{X}_1/\mathtt{t}_1, \ldots, \mathtt{X}_n/\mathtt{t}_n\}$, maps each logvar $\mathtt{X}_i$ to $\mathtt{t}_i$, where $\mathtt{t}_i$ is a logvar or a constant. A *grounding substitution* θ for an expression (e.g., an atom or a set of logvars) maps each logvar occurring in that expression to a constant. The set of all grounding substitutions for an expression E is denoted $grsub(\mathtt{E})$. The result of applying a substitution to an atom a is denoted $\mathtt{a}\theta$.

Given a relational database, we can built a number of *rules* that define its content, but were not explicitly represented in the database. The rules in logic programming can be expressed as *normal clauses* of the form $\mathtt{H} \leftarrow \mathtt{L_1,L_2,\ldots,L_n}$ where H is an atom and $\mathtt{L_1,L_2,\ldots,L_n}$ are literals. The rules can also be expressed with *definite clauses*, which are normal clauses containing no negated literals.

Example 8. *Based on the relations STUDENT and RECORD in Table 2.3 we can build the following rule*

$$\mathtt{grade(S,C,low)} \leftarrow \mathtt{difficulty(C,medium)}$$

which states that the grade of a student for a course is "low" if the difficulty of the course is "medium".

Clause $\mathtt{H} \leftarrow$ has an empty body. It represents an assertion or a *fact* and is used to specify the tuples of the relational database. The clause $\leftarrow L_1, L_2, \ldots, L_n$ represents a goal or a *query*. In the following example we illustrate how we can represent a relational database with a set of facts.

Example 9. *Relational database in Table 2.3 has three possible entities: students, courses and professors. Tables containing a list of instances of these entities are contained in tables STUDENT, COURSE and PROFESSOR, respectively. We can start the conversion by first listing the domain with the following facts:*

```
student(bob).     course(math).
student(ann).     course(bio).
student(mary).    course(chemistry).
student(jerry).   professor(john).
                  professor(susan).
```

Next, we represent the attributes of entities. For each entity attribute we create a predicate. Then for our relational data we would have:

```
intelligence(bob,105.0).      difficulty(chemistry,high).
intelligence(ann,115.0).      numHours(math,140.0).
intelligence(mary,107.0).     numHours(biology,90.0).
intelligence(jerry,112.0).    numHours(chemistry,100.0).
difficulty(math,high).        ability(john,high).
difficulty(biology,medium).   ability(susan,medium).
```

Finally, we convert relationships presented in tables TEACHES, RECORD and FRIENDS. Note that we can split the RECORD table in two relationships, one for *grade and one for satisfaction. However, the question is what to do with the primary key of the RECORD table because it only appears in this particular table and it is not a foreign key anywhere else. The question is whether we really need to model recordID for this particular problem. If we assume that there is at most one record a student has for a course, we can then combine student and course to a unique ID for this table. Note that we denote whether a student has a record for a course with predicate* `takes/2`*. Hence, we will have the following facts:*

```
teaches(john,biology).     grade(ann,biology,high).
teaches(john,chemistry).   grade(ann,math,high).
teaches(susan,math).       satisfaction(bob,biology,medium).
takes(bob,biology)).       satisfaction(bob,math,high).
takes(bob,math).           satisfaction(ann,biology,high).
takes(ann,biology).        satisfaction(ann,math,medium).
takes(ann,math).           friends(bob,ann).
grade(bob,biology,low).    friends(bob,mary).
grade(bob,math,medium).    friends(ann,jerry).
```

Note that not all the facts are listed. For example, `takes(bob,chemistry)` *is not listed because it has value false. By making the closed-world assumption we assume that everything that was not listed in the database is false or non-existing. For example, if a student does not take a course, then she does not have a grade for it.*

2.3.3 Relational Queries or Patterns

A query represents a method for extracting relevant information from the database. To put it differently, a query is a *pattern*, and one might be interested in finding whether or how many times the pattern occurs in the database. For example, one can count the instantiations of a graph pattern in graph data and thus obtain the sufficient statistics for performing parameter estimation (see Chapter 3). When doing structure learning, the focus is more on how to construct the space of queries in order to extract the patterns of interest. In this section we will give different representations of patterns and how these queries are answered in order to obtain the satisfying instances.

In database systems, queries are answered by using a special data manipulation languages such as the *structured query language (SQL)* (Date and Darwen 1997). Query in SQL makes use of the declarative *SELECT* statement which retrieves data from one or more tables, or expressions. Relational features can thus be represented as SQL queries (Popescul and Ungar 2003). In logic programming the queries are answered by building chains of deductions that combine rules and factual information, in order to prove or refute the validity of the initial query. In this dissertation we use *Prolog* (Bratko 2001), a logic programming language for representing relational data and expressing relational queries. Logically, the Prolog engine tries to find a resolution refutation of the negated query with the mechanism called *SLD resolution*. If the negated query can be refuted, it follows that the query, with the appropriate substitution, is a logical consequence of the program. In that case, all generated variable bindings are reported to the user, and the query is said to have succeeded.

Example 10. *For example consider the simple database in the Example 9 and a query expressed in Prolog:*

```
:- grade(bob,C,Value), satisfaction(bob,C,high)
```

With this query we ask for all the grade and course tuples of `bob` *for which he has high satisfaction. The substitutions that satisfy this query are* $\theta = \{\texttt{C/math}, \texttt{Value/medium}\}$*. In case we are interested in only the values of the grades in this query we can use a special-purpose predicate in Prolog: findall$(Template, Condition, Bag)$ in the following way:*

```
findall(Value,(grade(bob,C,Value),satisfaction(bob,C,high)),Bag)
```

with which we collect all `Value` *to the* `Bag` *for which the* `Condition` *holds.*

We would obtain the result to this query from the table RECORD with the following SQL-based query:

SELECT grade FROM RECORD WHERE student=bob, satisfaction=high

2.4 Modelling Probabilistic Relational Data

In Section 2.2 we discussed probabilistic graphical models, the graph-based propositional representations that encode probability distributions over variables, but are not able to capture the relational patterns that hold between them. Consequently, in the previous section we discussed representations based on first-order logic that are able to represent relations, but not probabilities. In this section we discuss statistical relational learning (SRL), which combines notions from relational representations and uncertainty. We present relational dependency networks in more detail, which is the SRL formalism we will use as the basis for work in this dissertation.

2.4.1 Statistical Relational Learning

Statistical relational learning (Getoor and Taskar 2007; De Raedt and Kersting 2011; De Raedt, Kersting, et al. 2016) or probabilistic logic programming is a subfield of machine learning and data mining that deals with relational domains where observations may be missing or noisy.

There exist two streams of research in SRL. One is to extend logic programming with probabilities and thus staying as close as possible to the logic formalisms (De Raedt, Kimmig, et al. 2007; Sato and Kameya 1997). The other one is to "upgrade" the existing probabilistic models with relational and logic-based elements (Kersting, De Raedt, and Kramer 2000; Neville and D. Jensen 2007; Fierens, Blockeel, Bruynooghe, et al. 2005; Richardson and Domingos 2006). In this dissertation we rely on an SRL formalism that represents the latter approach, and we introduce it in the next section.

2.4.2 Relational Dependency Networks

Relational dependency networks (RDNs) (Neville and D. Jensen 2007) are an SRL formalism that upgrades dependency networks, which we introduced in Section 2.2.2, to relational domains.

Representation

There are several ways to define RDNs, but we use a definition that uses first-order logic as a template language for constructing propositional dependency networks. In Section 2.3.2 we briefly reviewed the relevant concepts from a datalog subset of first-order logic that will be used in this section to define the syntax of RDNs.

Similar to LBNs (Fierens, Blockeel, Bruynooghe, et al. 2005), we use a set of statements or *random variable declarations* to define the random variables in a domain:

$$random(\mathtt{H}) \leftarrow L_1, \ldots, L_n$$

where H is an atom, and $L_1, \ldots, L_n$ is a conjunction of literals. Given a set of random variable declarations RVD, the set of random variables Φ is the set of all ground atoms $\mathtt{H}\theta$ for which there is a random variable declaration $random(\mathtt{H}) \leftarrow L_1 \ldots L_n$ in RVD and a substitution θ such that $L_1\theta, \ldots, L_n\theta$ is true given the background knowledge (amongst others specifying which ground atoms of the predicates in the body of the random variable declaration rules are true).

Example 11. *The random variable declaration for the atom* takes(S,C)

$$random(\mathtt{takes(S,C)}) \leftarrow \mathtt{student(S),\ course(C)} \tag{2.10}$$

creates one randvar for each student S *and course* C *in the domain.*

It must always be possible to evaluate the conjunction in the right-hand side of a random variable declaration, and we will use a closed-world assumption to guarantee this. As is common practice in many other probabilistic logical model frameworks (Fierens, Blockeel, Bruynooghe, et al. 2005; Richardson and Domingos 2006; Getoor, Friedman, et al. 2001), our random variable declarations specify all random variables that are potentially of interest. We ensure this with *relevancy conditions* $\mathtt{H} = not\_relevant \Leftrightarrow \xi$, which are a part of the background knowledge. More precisely, a special value *not_relevant* is assigned to the atom H if the constraint ξ is satisfied. We illustrate this in the following example.

Example 12. *The random variable declaration*

$$random(\mathtt{grade(S,C)}) \leftarrow \mathtt{student(S),course(C)} \tag{2.11}$$

specifies that every student gets a grade for every course, even though a precondition for obtaining a grade is that student S *must take course* C*. In this case,* grade(S,C) *would have a special value not_relevant in its domain, and we would have the background knowledge*

$$\mathtt{grade(S,C)} = not\_relevant \Leftrightarrow \mathtt{takes(S,C)} = false \tag{2.12}$$

Later, when learning the conditional dependency for `grade(S,C)` *on* `takes(S,C)` *and other random variables, we can easily use such hard background knowledge and reduce the learning problem to the subspace of the values of the parent random variables for which the dependent random variable is relevant.*

Let $\mathtt{H}\theta$ be a random variable. Given background knowledge, an interpretation I assigns a value to $\mathtt{H}\theta$ from its range or it assigns the special value *not_relevant* iff there exists a relevancy condition $\mathtt{H} \Leftrightarrow \xi$ in the background knowledge and $\xi\theta$ is true in I. The set of all groundings of a predicate P that have an assigned value $v \neq \mathit{not\_relevant}$ in interpretation I is denoted as $gr(\mathtt{P})^I$. We refer to the randvars in $gr(\mathtt{P})^I$ as P's *relevant* randvars.

Now, we will introduce relational features. For this, we first need to define aggregation functions.

Definition 4. (Aggregation Function) *An aggregation function for a domain D is a function that maps every finite multiset of elements from D to a single value from a range R.*

For example, *mode* is an aggregation function that maps a multiset of values from D to the most frequently occurring value in the multiset.

Definition 5. (Discrete Relational Feature) *Let $\mathcal{L}$ be a set of logvars,* C *be a conjunction of randvar-value tests of the form* $\mathtt{G} = v$ *where* G *is an atom and* $v \in range(\mathtt{G})$*,* A *be an atom, and* agg *be an aggregation function taking as input multisets of elements of* $range(\mathtt{A})$*. Assume the ranges of* A*, all atoms in both* C *and* agg *are discrete. Then, a* discrete relational feature $\mathtt{agg}_{\mathcal{L}}(\mathtt{A},\mathtt{C})^{(\theta,I)}$ *is a function that given any* $\theta \in \mathrm{grsub}(\mathcal{L})$ *and interpretation I maps* A *and* C *to*

$$\mathtt{agg}_{\mathcal{L}}(\mathtt{A},\mathtt{C})^{(\theta,I)} = \mathtt{agg}\left(\{I(\mathtt{A}\theta\theta') \mid \theta' \in \mathrm{grsub}(\mathtt{A}\theta,\mathtt{C}\theta) \textit{ and } \mathtt{C}\theta\theta' \textit{ holds in } I\ \}\right)$$

where we say $\mathtt{C}\theta\theta'$ *holds in I iff* $\forall(\mathtt{G} = v) \in C, I(\mathtt{G}\theta\theta') = v$.

A feature's range is the range of its aggregation function agg. The length of a feature is equal to the number of randvar-value tests in C plus one (for A). We will often omit the specification of the substitution θ and the interpretation I. Also, for brevity we will in some situations denote relational features as f, and its calculation for a specific substitution and interpretation as $f(\theta)$. We will now illustrate how a relational feature is calculated on an a small interpretation.

Example 13. *Consider the following interpretation representing a subset of the University example in which we for brevity assume that a grade exist for a student if he or she takes the course. Hence, we do not include assignments for the* `takes` *predicate.*

```
student(pete).             difficulty(bio,low).
student(mary).             difficulty(physics,high).
grade(pete,bio,low).       difficulty(math,high).
grade(pete,math,low).      difficulty(chem,med).
grade(pete,chem,high).
```

Next, consider the following relational feature:

$$\mathtt{agg}_{\{\mathtt{S}\}}(\mathtt{difficulty(C),grade(S,C)=low})$$

The following components of this relational feature can be recognized from Definition 5: a) a set of logvars $\mathcal{L}$ is $\{\mathtt{S}\}$ ranging over students, b) the conjunction `C` *consists of* `grade(S,C)=low` *and tests if a grade of a student in a course is low, c)* `A` *is* `difficulty(C)`.

Now assume we want to calculate this feature for the substitution $\theta = \{\mathtt{S/pete}\} \in \{\mathtt{S/pete}, \mathtt{S/mary}\}$, and `mode` *aggregation. This means that for student* `pete` *we will calculate the mode of the course difficulties in which he got a low grade:*

$$\begin{aligned}&\mathtt{mode}(\mathtt{difficulty(C),grade(S,C)=low}) = \\ &\quad = \mathtt{mode}\{I(\mathtt{difficulty(bio)}), I(\mathtt{difficulty(math)})\} \\ &\quad = \mathtt{mode}\{\mathtt{low,high}\}\end{aligned}$$

Note that the set of logvars $\mathcal{L}$ serves to denote what to aggregate over and can introduce different semantics of relational features. We will illustrate this in the following example.

Example 14. *Consider a simple relational feature:*

$$\mathtt{agg}_{\mathcal{L}}(\emptyset, \mathtt{grade(S,C)=low})^{(\theta,I)}$$

where $\emptyset$ denotes the non existence of an element (in this case an atom). Hence this feature only has one logical condition. Let `agg` *be a counting function* `count` *which simply returns the cardinality of a set. There are the two following cases when calculating the feature w.r.t. a given logvar set $\mathcal{L}$:*

1. *When $\mathcal{L} = \{\mathtt{S}\}$ and $\theta = \{\mathtt{S}/some\_student\}$ this feature represents how many low grades* `some_student` *has in interpretation I.*
2. *When $\mathcal{L} = \{\mathtt{C}\}$ and $\theta = \{\mathtt{C}/some\_course\}$ this feature represents how many students have low grades in* `some_course` *in interpretation I.*

There are also some extreme cases that can occur when calculating relational features. The following two cases for grounding a relational feature warrant mention:

1. $|\{I(\mathtt{A}\theta\theta') \mid \theta' \in grsub(\mathtt{A}\theta, \mathtt{C}\theta) \textit{ and } \mathtt{C}\theta\theta' \textit{ holds in } I\ \}| = 1$, for all $\theta \in grsub(\mathcal{L})$
2. $|\{I(\mathtt{A}\theta\theta') \mid \theta' \in grsub(\mathtt{A}\theta, \mathtt{C}\theta) \textit{ and } \mathtt{C}\theta\theta' \textit{ holds in } I\ \}| = 0$, for all $\theta \in grsub(\mathcal{L})$

The first case uses `value` (for brevity also denoted with $\simeq$ in Chapter 6) for the identity function which returns $I(\mathtt{A}\theta\theta')$. For example, if each student `S` has exactly one value for intelligence, then the relational feature $value_{\{\mathtt{S}\}}(\mathtt{intelligence(S)}, \emptyset)$ simply returns the value taken by the randvar `intelligence(S)`, which represents the `intelligence` of a student `S`, in interpretation I. The second case requires applying an aggregation function to the empty set. Some aggregation functions (e.g., mode) are not defined on the empty set, and in this case $\mathtt{agg}_{\mathcal{L}}(\mathtt{A}, \mathtt{C})$ returns the value $undefined$.

Definition 6. (Discrete Dependency Statement) *A* discrete dependency statement *is of the form* `G` | *Parents*(`G`)*.* `G` *is the* target *atom that has a discrete range and whose arguments are all logvars. Parents*(`G`) *is a set of discrete relational features, where for each* $\mathtt{agg}(\mathtt{A}, \mathtt{C}) \in$ *Parents*(`G`)*, $\mathcal{L}$ is a subset of the logvars in* `G`*, where $\mathcal{L}$ is the union of logvars in* `A` *and* `C`*. Each dependency statement has an associated conditional probability distribution (CPD) which quantifies how the target atom depends on its parent set.*

Example 15. *An example of a discrete dependency statement is:*

$$\mathtt{intelligence(S)} \mid \mathtt{mode}_{\{\mathtt{S}\}}(\mathtt{grade(S,C)}, \mathtt{takes(S,C)=true})$$

which states that a student's intelligence depends on the mode *of grades received across all courses the student has taken. As each student can take a varying number of courses, an aggregation function, such as* mode *in this example, is needed to combine the values from the varying number of parents into a single value.*

We are now ready to formally define an RDN:

Definition 7. (RDN) *An RDN is a tuple $(\mathcal{P}, RVD, dep)$, where $\mathcal{P}$ is a set of predicates, each with a discrete range, RVD is a set of randvar declarations, and dep is a function that maps each $\mathtt{P} \in \mathcal{P}$ to a discrete dependency statement.*

An RDN $(\mathcal{P}, RVD, dep)$ is a template for constructing propositional DNs. Given the background knowledge and a set of randvar declarations RVD, an induced DN has a node for each randvar $\mathtt{G}\theta \in \Phi$.

The parent set of a ground atom $\mathtt{G}\theta$ in a dependency network is defined as

$$Parents(\mathtt{G}\theta) = Parents_A(\mathtt{G}\theta) \cup Parents_C(\mathtt{G}\theta)$$

where

$$\begin{aligned} Parents_A(\mathtt{G}\theta) &= \left\{A\theta\theta' \mid \exists\, \mathtt{agg}_{\mathcal{L}}(\mathtt{A},\mathtt{C}) \in Parents(\mathtt{G}) : \theta' \in grsub((C\theta, A\theta))\right\} \\ Parents_C(\mathtt{G}\theta) &= \cup \left\{C\theta\theta' \mid \exists\, \mathtt{agg}_{\mathcal{L}}(\mathtt{A},\mathtt{C}) \in Parents(\mathtt{G}) : \theta' \in grsub((C\theta, A\theta))\right\} \end{aligned} \tag{2.13}$$

There is an arc between two ground atoms $\mathtt{G}\theta$ and $\mathtt{G}'\theta$, if $\mathtt{G}'\theta \in Parents(\mathtt{G}\theta)$. The CPDs are shared across all randvars that originate from the same predicate.

The pseudo-loglikelihood of an RDN M for an interpretation I involves only the relevant randvars and it is calculated as:

$$PLL(M; I) = \sum_{\mathtt{P} \in \mathcal{P}} \sum_{g \in gr(\mathtt{P})^I} \log\left[p(I(\mathtt{g}) \mid I(Parents(\mathtt{g}))\right]. \tag{2.14}$$

Example 16. *Consider the following simple RDN for a domain with the following randvar declarations:*

$$\mathtt{random(intelligence(S))} \leftarrow \mathtt{student(S)}$$

$$\mathtt{random(takes(S,C))} \leftarrow \mathtt{student(S), course(C)}$$

$$\mathtt{random(grade(S,C))} \leftarrow \mathtt{student(S), course(C)}$$

$$\mathtt{random(difficulty(C))} \leftarrow \mathtt{course(C)}$$

where each predicate has a discrete range and the following dependency statement:

$$\texttt{grade(S,C)} \;\Big|\; \begin{array}{l}\texttt{value}(\texttt{intelligence(S)},\emptyset),\\ \texttt{value}(\texttt{difficulty(C)},\emptyset)\end{array}$$

The dependency states that a student's grade in a course depends on the student's intelligence and the difficulty of the course. Note that this statement says that all ways of instantiating the logvars `S` *and* `C` *have an identical probabilistic relationship with* `S`*'s intelligence and* `C`*'s difficulty. Figure 2.3 shows an induced propositional DN for this RDN given the relevancy condition on* `grade/2` *specified in (2.12), and the domain introduced in Example 9 with two students* `bob` *and* `ann`*, and two courses* `math` *and* `bio`*. The dashed arrows denote the relevancy conditions for the* `grade/2` *randvars.*

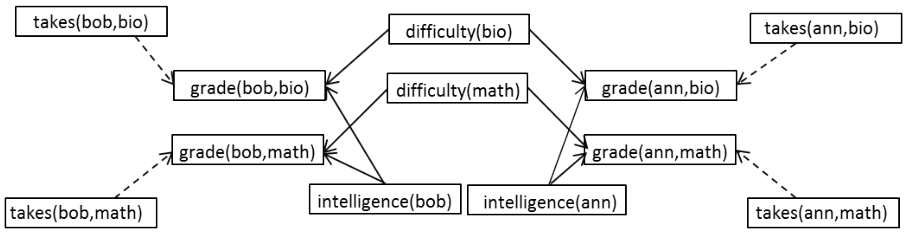

Figure 2.3: The DN induced by grounding the RDN specified in Example 16. The dashed arrows specify the relevancy condition on `grade/2`.

Inference

Given that RDNs are templates for constructing DNs, they inherent the semantics of DNs (Neville and D. Jensen 2007). Namely, a consistent RDN specifies a joint probability distribution over the randvars of a relational data set. Similarly, a unique joint probability distribution for an RDN can be obtained by grounding out the model to obtain a DN and then running an ordered pseudo-Gibbs sampler on the DN. Again, this can be done regardless of whether the model is consistent. The distribution of an inconsistent RDN is the stationary distribution of an ordered pseudo-Gibbs sampler (if it exists) applied to the model.

Learning

Learning the structure of an RDN follows the same paradigm as in the propositional case we described in Section 2.2.2: the CPD for each predicate is learned in turn. Even though it is possible to model the CPDs as a some form of a CPT, this is normally done by learning a relational probability tree (Neville and D. Jensen 2007) or relational regression tree for each predicate. Next we briefly introduce these models.

2.4.3 Relational Conditional Probability Models

The CPDs of relational probabilistic models can be modelled in several ways. One simple approach is to use a relational version of conditional probability tables. We already discussed in the propositional case in Section 2.2.2 why this is an inefficient representation. The same applies to relational domains. Next, we present relational decision trees as an elegant way of modelling conditional probabilities.

Relational Decision Trees

Relational decision trees or first-order logical decision trees (Blockeel and De Raedt 1998) are binary decision trees where (1) the nodes of the tree contain a conjunction of literals, and (2) different nodes may share variables, under the following restriction: a variable that is introduced in a node (which means that it does not occur in higher nodes) must not occur in the right branch of that node. They are essentially an upgrade of decision trees, which we briefly mentioned in Section 2.2, to relational domains. Unlike propositional decision trees, relational decision trees envision the main principles of relational data modelling such as non-i.i.d. entities and heterogeneity.

Learning relational decision trees is in a sense similar to learning decision trees. The point where relational decision trees differ significantly from propositional trees is in the format of the tests that can serve as internal nodes: for the former the test are propositions, while for the latter the tests are relational features. When learning first-order logic decision trees the feature space is obtained with a refinement operator under *θ-subsumption* (Muggleton and De Raedt 1994; Plotkin 1970). Such an operator ρ maps clauses onto sets of clauses, such that for any clause c and $\forall c' \in \rho(c)$, c θ-subsumes c'. A clause c_1 θ-subsumes another clause c_2 if and only if there is a variable substitution θ such that $c_1\theta \subseteq c_2$. The tree is learned recursively in the following way. In order to refine a node with another relational feature f the data is split into two sets: where f is true and where f is false. The feature f is chosen as a refinement if it improves some scoring function, such as likelihood, that indicates how good the tree is. If the feature f is chosen, the algorithm adds it as an internal test and the procedure continues by refining the feature f. The heterogeneous aspect of relational domains is usually handled with aggregation (Van Assche et al. 2006).

Depending on whether the range of the target attribute is discrete or numeric, we have *classification* or *regression* trees. In order to model conditional probability distributions we naturally use regression trees and there are several existing representations for this:

- **Relational probability trees (RPTs)** (Neville, D. Jensen, et al. 2003) are used when the target attribute is discrete and we are interested in modelling a probability mass function for that attribute. Sorting down an example in a relational probability tree gives the probability distribution for the target attribute for that branch.
- **Relational model trees** (Vens et al. 2006) are regression trees used to model non-trivial functions such as linear and logistic regression between the target attribute and internal attributes. This tree can be used to model probability distributions (densities) for discrete (numeric) attributes. For example, while logistic regression gives the probability of a discrete variable, linear regression can provide a probability density for a numeric attribute.

2.5 Dynamic Relational Models

In the previous sections we discussed possible ways of modelling propositional and relational data. The environment of the agent typically contains varying numbers of objects with *evolving* relations among them, and one needs relational representations to generalize over specific types of objects in order to make the modelling and learning feasible (Lang, Toussaint, and Kersting 2012). Representations that we covered in the previous sections were all *static* which means that they do not take into account the changing nature of the underlying processes. In this section, we will discuss how to represent *dynamic* relational domains. For that discussion we will briefly discuss Markov decision processes (MDP) and discuss their upgrade to relational domains by means of distributional clauses (DCs). Naturally, dynamic models are very important in robotic applications where one is interested into performing control and action model learning under uncertainty.

2.5.1 Markov Decision Processes

The problem of planning under uncertainty can be modeled as a Markov decision process (MDP). In an MDP, an agent interacts with its environment, described using a set of *states* S, a set of *actions* $\mathcal{A}$ that the agent can perform, a *transition function* $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, and a *reward function* $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. That is, when in state s_t and performing action a_t, the probability of reaching s_{t+1} is given by $p(s_{t+1}|s_t, a_t)$, for which the agent receives the reward $R(s_t, a_t)$. It is assumed that the agent operates over a finite number of time steps $t = 0, 1, \dots, T$, with the goal of maximizing the expected reward: $\mathbb{E}[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)]$, where s_0 is the start state, a_0 the first action, and $\gamma \in [0, 1]$ is a discount factor. In this thesis we consider only goal-oriented MDPs where there is a goal $g \subset \mathcal{S}$ to reach, and the reward is high if we reach the goal, and low otherwise.

2.5.2 Distributional Clauses

Distributional clauses (Gutmann, Thon, et al. 2011) are a formalism that upgrades MDPs to relational domains by representing each state of the world as a set of related objects and properties. Thus, to specify them we rely on the methods for modelling relational data, such as the variation to first-order logic, we introduced in Section 2.3.2, and elements of statistical relational learning we introduced in Section 2.4.

Formally, a *distributional clause* is a formula of the form $\mathtt{H} \sim \mathcal{D} \leftarrow \mathtt{L_1}, \dots, \mathtt{L_n}$, where the $\mathtt{L_i}$ are literals and $\sim$ is a binary predicate written in infix notation. The intended meaning of a distributional clause is that each ground instance of the clause $(\mathtt{H} \sim \mathcal{D} \leftarrow \mathtt{L_1}, \dots, \mathtt{L_n})\theta$ defines the random variable $\mathtt{H}\theta$ with distribution $\mathcal{D}\theta$ whenever all the $\mathtt{L_i}\theta$ are true, where θ is a substitution. In distributional clauses (DCs) (Gutmann, Thon, et al. 2011; Nitti, De Laet, et al. 2013), an interpretation I assigns a value $I(\mathtt{G}\theta)$ to each ground atom $\mathtt{G}\theta$. While in logic programming that value will be true or false, in DCs the values can also be discrete or numeric as ground atoms represent random variables. A distributional clause is a template to define conditional probabilities: $p(\mathtt{H}\theta|(\mathtt{L_1},\ldots,\mathtt{L_n})\theta) = \mathcal{D}\theta$. The term $\mathcal{D}$ can be nonground, i.e., values, probabilities, or distribution parameters can be related to conditions in the body. Furthermore, given a random variable $\mathtt{r}$, the term $\simeq(\mathtt{r})$ constructed from the reserved functor $\simeq/1$ represents the value of $\mathtt{r}$. A *distributional program* is a set of distributional clauses (some of which may be deterministic) that defines a distribution over possible worlds, which in turn defines the underlying semantics.

Dynamic distributional clauses (DDC) (Nitti, Belle, et al. 2015) associate a time index to each random variable to capture temporal information and also support continuous distributions. It is straightforward to specify an MDP using DDC and we will illustrate that with the following example.

Example 17. *Let us consider a scenario of an object manipulation. There is an object on the table and the robot has to move it to a given region. This scenario is modeled with the following DDC representing the MDP:*

$$\mathtt{pos(ID)_{t+1}} \sim \mathtt{gaussian}(\simeq(\mathtt{pos(ID)_t}) + (\mathtt{DX},\mathtt{DY}),\Sigma) \leftarrow$$
$$\mathtt{push(ID,(DX,DY))}. \tag{2.15}$$

$$\mathtt{stop_t} \leftarrow \mathtt{dist}(\simeq(\mathtt{pos(ID)_t}),(0.6,1.0)) < 0.1. \tag{2.16}$$

$$\mathtt{reward(100)_t} \leftarrow \mathtt{stop_t}. \tag{2.17}$$

$$\mathtt{reward(-1)_t} \leftarrow \mathtt{not(stop_t)}. \tag{2.18}$$

The DDC clause (2.15) defines the state transition model, i.e. the next position $\mathtt{pos(ID)_{t+1}}$ *of an object* $\mathtt{ID}$ *after a push action with displacement* $(\mathtt{DX},\mathtt{DY})$*. The deterministic clause (2.16) defines when the goal is reached (e.g., an object is close to point* $(0.6, 1.0)$*) and the remaining clauses define the reward function. Note that* $\mathtt{pos(1)_{t+1}}$ *represents a random variable, i.e., the position of object* 1 *at time* $t+1$*, with predicate-like notation. But unlike standard relational representations, a random variable can have a continuous (or categorical) range. Thus, in DC and DDC an interpretation assigns each random variable a value in its range.*

Static inference in DC is performed using importance sampling, while filtering in dynamic models is performed using particle filtering methods (Nitti, De Laet, et al. 2013).

3 Graph Sampling for Efficient Parameter Estimation in SRL

3.1 Introduction

Recall from Section 2.4 that estimating the parameters of a model often involves *counting* the number of times a configuration of a subset of the variables occurs in the data. Furthermore, in computationally demanding tasks such as structure learning in SRL, statistics for many different patterns must be repeatedly collected during the learning process. The task of counting how many times a pattern is *embedded* into a graph has been extensively explored in the graph mining community, which often focuses on large patterns and graphs. Because relational data and patterns can be represented as *graphs*, we could translate some of the efficient approaches from the graph mining community to SRL.

Unfortunately, in general, counting the number of occurrences of a graph pattern in a network is #P-complete (Valiant 1979). To address this shortcoming, we introduce in this chapter a new approach to count the number of embeddings of a pattern in a graph based on a fully *polynomial* randomized *approximation* scheme suggested by Fürer and Kasiviswanathan (2008). For practical use in a data mining context, this theoretical approach has several drawbacks as well as several crucial open questions. First, it assumes that an input pattern has an *ordered bipartite decomposition (OBD)* and that this decomposition is given. This assumption decompositions a large pattern in smaller feasible graphs such that embeddings can be found in a stage-wise manner, i.e., for each decomposition independently. However, some interesting patterns might not have an OBD, and it would be interesting to examine how the algorithm performs when this condition is violated. Second, it assumes that all graphs are unlabeled and undirected, which limits the applicability of the approach. Finally, only the number of pattern embeddings is estimated, whereas in order to show that the approach is beneficial for a wider range of pattern mining tasks, it would be interesting to assess the accuracy of the statistics derived from the estimated counts.

3.1.1 Contributions and Bibliographical Note

The main contributions in this chapter are the following. First, we propose an algorithm, based on the theoretical work of Fürer and Kasiviswanathan (2008), for approximately counting embeddings into random graphs. Second, we propose a heuristic approach to find the OBD of a pattern, which is assumed to be given in the original work. Third, we use our approach to perform parameter estimation for logical Bayesian networks (Fierens, Blockeel, Bruynooghe, et al. 2005), an SRL formalism that serves as a first-order logic based template for constructing Bayesian networks, and we compare our proposed approach with two baseline approaches. Fourth, while Fürer and Kasiviswanathan (2008) provide only an error bound, we also estimate the accuracy of the computed statistics from the sample. Finally, whereas the original work assumes random domain graphs, we experimentally assess the algorithm also on *power law* graphs to verify whether the guarantees of the algorithm will hold.

This work is under the revision as:

> *Irma Ravkic, Martin Znidarsic, Jan Ramon, Jesse Davis (2016) "Graph sampling for efficient parameter estimation in statistical relational learning" (under revision for Data Mining and Knowledge Discovery Journal via the ECML PKDD journal track)*

The contributions of this work are divided across the first two authors of the paper mentioned above, in the following way:

- Martin Znidarsic implemented the three tested algorithms for counting pattern embeddings: our proposed approach, an exact approach, and a random sampling approach. He also proposed and evaluated two algorithms for obtaining the ordered bipartite decomposition.
- The author of this dissertation built upon the implementation of Martin Znidarsic by adding a number of modifications and improvements in order to allow parallelization and cyclic graphs. She also developed an extensive experimental evaluation which involved generating a large space of pattern graphs, and sampling from this space a subset of interesting graphs for which a number of statistics is collected in order to evaluate the tested approaches.

3.1.2 Chapter Structure

The remainder of this chapter is structured as follows. First, we provide a short background on graph theory and we introduce and illustrate the ordered bipartite decomposition. Second, we define the problem of counting the number of embeddings in graphs and introduce two baseline approaches we will use in our experimental setup. Then we introduce the proposed sampling approach of Fürer and Kasiviswanathan. Next we present two algorithms for obtaining an ordered bipartite decomposition (OBD) of a pattern needed for the theoretical guarantees of the proposed approach to hold. This is followed by a brief overview of possible statistics that can be extracted from the collected embeddings. Then we present our experimental methodology and results, provide a discussion, and conclude the chapter.

3.2 Graphs

A labeled graph is a tuple $G = (V, E, \Sigma, \lambda)$, where V is a set of vertices (also called nodes), $E \subseteq \big\{\{u, v\} \mid u, v \in V\big\}$ is a set of edges, Σ is a set of labels and $\lambda : (V \cup E) \to \Sigma$ is a function that assigns labels to nodes and edges. For a graph G, we refer to its set of vertices with $V(G)$, to its set of edges with $E(G)$, to its alphabet with Σ_G and to its labeling function with λ_G. For a graph G and a vertex $v \in V(G)$ the set of *neighbors* of v in G is $N_G(v) = \{u \in V(G) \mid \{u, v\} \in E(G)\}$. A graph H is a *subgraph* of a graph G if $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ and

$\forall x \in V(H) \cup E(H) : \lambda_H(x) = \lambda_G(x)$. The label set Σ can contain attribute-value pairs as well. More precisely, for a node v with label $attribute = value$ we denote the $value$ of v with $value(v)$. We also allow for wildcards # to be used as $value$. This means that the node can take on values, but we do not care in this particular case which value it is. We will often omit # in $attribute = \#$, and denote the label of the node with $attribute$.

Example 18. *Consider the labeled graph in Figure 3.1. The nodes in the graph denote persons (men or women) together with their friendship or marriage relations to other people and with their satisfaction with their salary. Note that labels for persons, salary and satisfaction are attribute-value pairs (e.g., satisfaction=high for mike).*

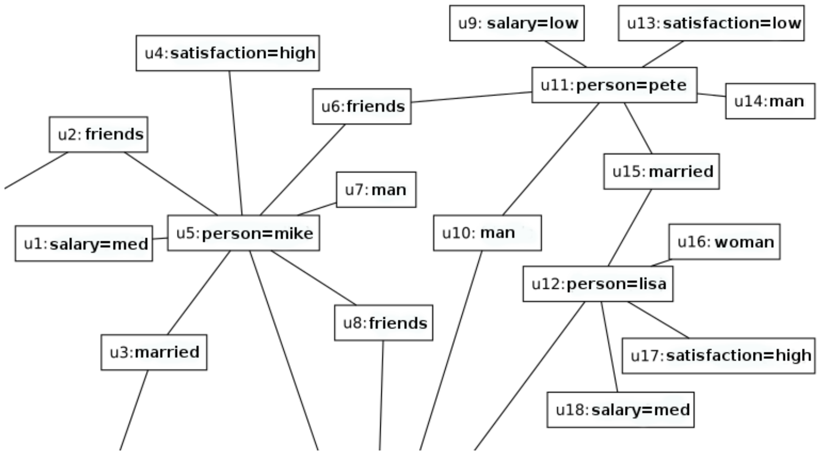

Figure 3.1: An example of a labeled domain graph depicting a number of *persons (men or women)* together with their *friendship* or *marriage* relations to other people and with their *satisfaction* with their *salary*.

3.3 Subgraph Matching

A relational query can be represented as a pattern graph such that attributes and variables of interest are nodes, and the edges represent the relations between the nodes (e.g., in (Neville and D. Jensen 2007; Das et al. 2016)). We first introduce some necessary definitions and notations.

A *pattern* or a *query* graph over a set of labels Σ is a graph P with $\Sigma_P \in \Sigma$ (i.e., the labels on vertices and edges of P are subsets of Σ). Given a pattern P and a domain graph D, we call an injection φ between nodes of P and nodes of D an *embedding* of P onto D if φ maps vertices (edges) $x \in P$ to vertices (edges) $\varphi(x) \in D$ such that $\lambda_D(\varphi(x)) \in \lambda_P(x)$. We will often use the term *image* for $\varphi(x)$ as in the standard mathematical sense: the subset of a function's codomain which is the output of the function from a subset of its domain. Note again that the labels of P which contain more than one element are wild cards allowing to match to several vertex/edge image labels. If G is a graph and $S \subseteq V(G)$, then $G[S]$ is the subgraph of G induced by S, i.e. $V(G[S]) = S$, $E(G[S]) = \{\{u,v\} \in E(G) | u,v \in S\}$, $\Sigma_{G[S]} = \Sigma_G$ and $\lambda_{G[S]} = \lambda_{G|V(G[S]) \cup E(G[S])}$.

There are multiple ways of performing the matching between two graphs. A *homomorphism* from a graph P to a graph D is a mapping $\varphi : V(P) \to V(D)$ such that (i) $\forall v \in V(P), \lambda_P(v) = \lambda_D(\varphi(v))$, and (ii) $\forall \{u,v\} \in E(P)$, $\{\varphi(u), \varphi(v)\} \in E(D) \wedge \lambda_P(\{u,v\}) = \lambda_D(\{\varphi(u), \varphi(v)\})$. Example 19 illustrates graph homomorphism for a simple pattern graph.

Example 19. *Consider a pattern graph P to be mapped by means of homomorphism to another graph D illustrated. These graphs are illustrated in Figure 3.2. The nodes of graph P and graph D are labeled with three kind of labels expressed as shapes of the nodes: {triangle, circle and square}. The homomorphism from graph P to graph D is:* $\varphi(a) = 0, \varphi(b) = 1, \varphi(c) = 2, \varphi(d) = 0, \varphi(e) = 1$. *Note that a non-edge in pattern P between nodes e and c is mapped to an edge in pattern D going from node* 1 *to node* 2.

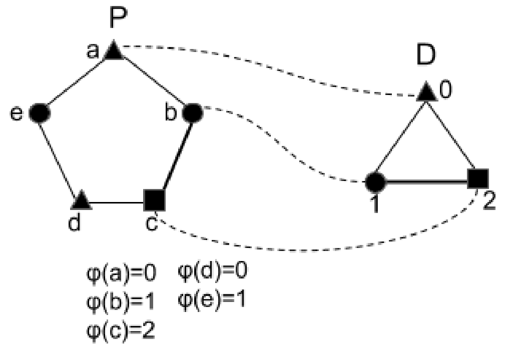

Figure 3.2: An example of a homomorphism between two graphs in which nodes are labeled with the shapes of the nodes: {triangle, circle, square}. The φ function represents the mapping under the homomorphism.

A *subgraph isomorphism* from P to D is a homomorphism φ from P to D such that $\forall u, v \in V(P), \varphi(u) \neq \varphi(v)$. Graph isomorphism is illustrated in Example 20.

Example 20. *Consider a pattern graph P to be mapped by means of isomorphism operator to another graph D shown in Figure 3.3. The nodes of graph P and graph D are labeled with three kind of labels expressed as shapes of the nodes: {triangle, circle and square}. The isomorphism from graph P to graph D is:* $\varphi(a) = 1, \varphi(b) = 3, \varphi(c) = 4, \varphi(d) = 2, \varphi(e) = 5$. *Note that each edge in pattern P is mapped to an edge in pattern D (e.g., edge between b and c is mapped to the edge between* 3 *and* 4*), and each non-edge is mapped to a non-edge (e.g., for a non-edge between b and e there is no edge between nodes* 3 *and* 5 *in D).*

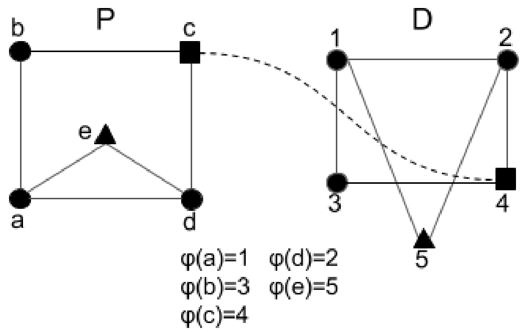

Figure 3.3: An example of an isomorphism between two graphs in which nodes are labeled with the shapes of the nodes: {triangle, circle, square}. The φ function represents the mapping under the isomorphism.

Whether homomorphism or subgraph isomorphism is more appropriate to use as a matching operator depends on the application. Our results are generic and apply to both cases. We use $\text{emb}(P, D)$ to refer to $\text{emb}_x(P, D)$ for any $x \in \{hom, iso\}$ and refer to its elements as *embeddings*. For a domain graph D and a pattern P, a *partial embedding* of P in D is an element of $emb(P[S], D)$ where $S \subseteq V(P)$.

Given an embedding φ of P in D and a set $S \subseteq V(P)$, we denote with $val_{S:D}(\varphi)$ the labels of the embedding for a subset of nodes. More formally, $val_{S:D}(\varphi) = \{(u, \lambda_D(v)) \mid u \in S \wedge (u, v) \in \varphi\}$. We define the multiset of values for $S \subseteq V(P)$ in graph D by $\mathcal{V}al(S : P, D) = \{val_{S:D}(\varphi))\}_{\varphi \in \text{emb}(P,D)}$. When $S = V(P)$, we define $val_D(\varphi) = val_{V(P):D}(\varphi)$, and $\mathcal{V}al(P, D) = \mathcal{V}al(V(P) : P, D)$.

We also denote the range of values for S of P in D by $\mathcal{R}ange(S : P, D))$ and define it as $\mathcal{R}ange(S : P, D)) = \{val_{S:D}(\varphi) \mid \varphi \in \text{emb}(P, D)\}$. The essential difference between $\mathcal{R}ange(S : P, D)$ and $\mathcal{V}al(S : P, D)$ is that the first is a normal set (containing each value once) while the second is a multiset possibly containing the same value several times. When $S = V(P)$ then $\mathcal{R}ange(P, D) = \mathcal{R}ange(V(P) : P, D)$.

We illustrate the introduced notations and definitions in Example 21.

Example 21. *Consider the domain graph in Figure 3.1. We will denote this graph with D. Also consider a simple pattern in Figure 3.4 we denote with P. There are two*

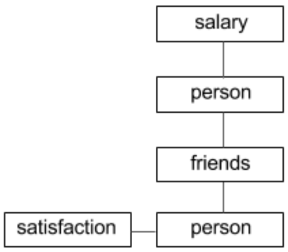

Figure 3.4: A pattern graph with satisfaction and salary attributes for two persons that are friends.

possible embeddings of this pattern in the data graph D:

$$\varphi_1 = \{(p_1, u_4), (p_2, u_5), (p_3, u_6), (p_4, u_{11}), (p_5, u_9)\}$$

$$\varphi_2 = \{(p_1, u_{13}), (p_2, u_{11}), (p_3, u_6), (p_4, u_5), (p_5, u_4)\}$$

Let $S = \{p_1, p_5\} \subseteq V(P)$ and let us consider embedding φ_1. Then $val_{S:D}(\varphi_1) = \{(p1, \lambda_D(u_4)), (p_5, \lambda_D(u_9)\} = \{(p1, satisfaction = high), (p_5, salary = low)\}$. The multiset of values for S is $\mathcal{V}al(S : P, D) = \{\{(p1, satisfaction = high), (p_5, salary = low)\}, \{(p1, satisfaction = low), (p_5, salary = med)\}\}$. $\mathcal{R}ange(P, D)$ is in this case equivalent to $\mathcal{V}al(S : P, D)$.

3.4 Ordered Bipartite Decomposition of Graphs

Decomposing graphs allows for a divide-and-conquer strategy to be applied for complex problems and it represents a preprocessing step for many graph theory techniques. In this thesis we use this strategy for obtaining the count of a pattern's embeddings in a graph. The general idea is to decompose the vertices involved in a complex pattern into multiple partitions, where each partition contains as few vertices as possible. The matching task then becomes simpler as only the vertices in a single partition need to be matched simultaneously, with the current partition's matching being conditioned on the assignment given to the vertices in previous partitions. Ordered bipartite decomposition (OBD) was introduced by Fürer and Kasiviswanathan (2008) and it is a crucial component of our approach. In this section we define bipartite graphs and OBD.

An *independent set* is a set of vertices in a graph, no two of which are adjacent. *Graph partition* is a problem of dividing vertices and edges of a graph into smaller components with specific properties. A *bipartite* graph $G = (U, V, E, \Sigma, \lambda)$ is a graph whose vertices can be divided into two disjoint independent sets U and V such that every edge in $e \in E$ connects a vertex in U to one in V. A bipartite graph can be seen as colouring the nodes in U to one colour, and nodes in V to another, and each edge has endpoints of different colours. An example of a bipartite graph is modelling of the relationship of football players with their clubs: each player is associated with a club. An example of a non-bipartite graph is a triangle: if we colour one node into red, another one in blue, the third node cannot be assigned neither red nor blue colour, as it will be connected to a node of the same colour as itself.

Definition 8 (Ordered bipartite decomposition (Fürer and Kasiviswanathan 2008)). *An ordered bipartite decomposition of a graph $P = (V_P, E_P)$ is a sequence $V_1, ..., V_l$ of subsets of V_P such that:*

1. *$V_1, ..., V_l$ form a partition of V_P*
2. *Each of the V_i (for $i \in [l] = 1, ..., l$) is an independent set in P*
3. *$\forall v \exists j$ such that $v \in V_i$ implies $N_P(v) \subseteq (\bigcup_{k<i} V_k) \cup V_j$*

Informally, an ordered bipartite decomposition is a labeling of vertices ensuring that every edge is between vertices having different labels and for every vertex all neighbors with a higher label in the order have identical labels. Fürer and Kasiviswanathan (2008) showed that many graph classes have such a decomposition, while at the same time many simple small graphs (e.g., a triangle) may not possess such a decomposition. The size of an OBD is the number of partition classes in it and the size of the largest partition class defines its *width*. We illustrate an OBD in Example 22.

Example 22. *Consider the two graphs in Figure 3.5. Graph (a) is a triangle graph. There exists a trivial decomposition* $\{v1\}, \{v2\}, \{v3\}$ *satisfying Properties* 1 *and* 2 *of Definition 8. However, it is not possible to satisfy Property 3 for this graph, hence, there is no OBD for it. The OBD for graph (b) is* $\{v1\}, \{v2\}, \{v3, v4\}$ *where each partition is labeled with, for example, I, II, and III. The reader can check that indeed all three properties hold. For example, for the vertex* $v2$ *labeled as II, all the neighbours (*$v3$ *and* $v4$*) are labelled identically with a higher label III.*

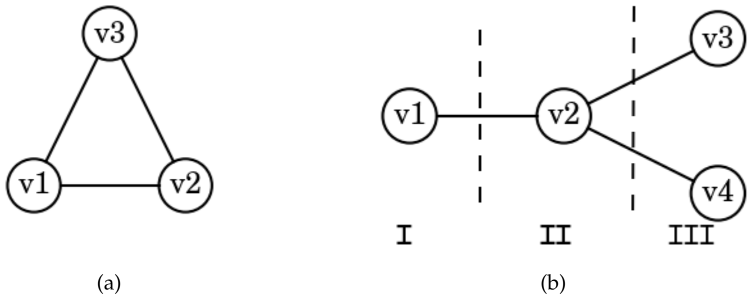

Figure 3.5: An example of a) a non-bipartite triangle graph for which there does not exist an OBD, b) a graph with $\{v1\}\{v2\}\{v3, v4\}$ OBD.

3.5 Counting the Number of Embeddings in Graphs

Section 3.3 discussed representing relational queries as graphs and we illustrated two matching operators most frequently used in graph mining: graph isomorphism and graph homomorphism. The problem of collecting statistics about the occurrences of a graph pattern in a network, irrespective of the matching operator, can be formalized as:

- **Given:** a domain graph D, a pattern P, and a statistic of interest f
- **Find:** $f(P, D)$

The basic procedure that is used in all the approaches in this chapter is to find all extensions to a partial embedding φ. For that purpose we use the procedure in Algorithm 1. The function receives a domain graph D, a pattern graph P, a partial embedding φ and a set of vertices $S \subseteq P$ for which we want to find the images. The function loops through all the vertices in $v \in S$ and tries to find images of v which are then stored in the C_v variable. It then constructs the set of all possible embeddings such that each embedding is the concatenation of the partial embedding φ and one of the combination of images found for vertices in S. The test $\varphi' \in Emb(P[\varphi'^{-1}(D)], D)$ checks if the edges and labels are preserved by φ'. If subgraph isomorphism is used as the matching operator (rather than homomorphism), one must additionally check that no two vertices receive the same image.

Algorithm 1 Extend partial embeddings.

```
1: function PartialExt($D$,$P$,$\varphi$,$S$)
2:     for all $v \in S$ do                    ▷ Loop through vertices in the partition
3:         if $\exists(x,y) \in \varphi : x \in N_P(v)$ then
4:             $C_v \leftarrow \{u \in N_D(y) \mid \lambda(u) \in \lambda(v)\}$
5:         else
6:             $C_v \leftarrow \{u \in V(D) \mid \lambda(u) \in \lambda(v)\}$
7:     return $\{\varphi' \mid \varphi' = \varphi \cup \{(v,u_v)\}_{v\in S} \wedge \forall v \in S : u_v \in C_v \wedge \varphi' \in Emb(P[\varphi'^{-1}(D)], D)\}$
```

Next, we introduce an exhaustive approach and a sampling approach for finding the number of embeddings of a pattern in a domain graph.

3.5.1 Exhaustive Approach

As the name suggests, this approach, outlined in Algorithm 2 finds all possible embeddings of a pattern. The algorithm receives a domain graph D, a pattern graph P, and a fixed ordering $\mathcal{O}$ on the pattern vertices as input. We use $\mathcal{O}_i$ to denote the i-th vertex in the ordering $\mathcal{O}$, that is, the i-th vertex that is assigned an image. Starting from an empty embedding, the algorithm works as follows. The PartialExt function defined in Algorithm 1 finds all extensions to a partial embedding φ by assigning an image to the $\mathcal{O}_{|\varphi|+1}$th vertex.

When choosing an ordering $\mathcal{O}$, a typical heuristic is to first select images for vertices that have a small number of possible images. We require that every vertex, except the first one, is adjacent to at least one other vertex which precedes it in the ordering. A large literature exists on heuristics and various forms of lookahead (Ullmann 1976), which is outside the scope of this thesis.

Algorithm 2 Exhaustive approach.

```
1: function AllEmb(graph $D$, pattern $P$, ordering $\mathcal{O}$)
2:     return AllEmb($D$, $P$, $\mathcal{O}$, $\{\}$)              ▷ Return a set of embeddings
3: function AllEmb($D$,$P$,$\mathcal{O}$,$\varphi$)
4:     if $|\varphi| = |V(P)|$ then                  ▷ If the pattern is fully embedded
5:         return $\{\varphi\}$
6:     else                        ▷ Repeat the procedure for each found extension
7:         return $\cup_{\varphi' \in \text{PartialExt}(D,P,\varphi,\{\mathcal{O}_{|\varphi|+1}\})} \text{AllEmb}(D, P, \mathcal{O}, \varphi')$
```

3.5.2 Random Vertex Sampling

Algorithm 3 presents an approximate algorithm for computing all embeddings of a pattern based on sampling. It receives the same input as the exhaustive approach (i.e., function AllEmb in Algorithm 2). At a high level, this algorithm samples the first part of the embedding randomly and then exhaustively computes all of its completions. This algorithm differs from the exhaustive algorithm in that it first finds an image for the first vertex $\mathcal{O}_1$ by randomly sampling, with replacement, from the set of vertices with a matching label instead of exhaustively iterating over all of them.

One can observe that every embedding has the same probability of being encountered by this algorithm. However, several embeddings may be included in the sample in the same iteration, and hence these may not be independent. Therefore, in line 7 we form a sample by a set of sets of embeddings (indicating these dependencies explicitly) rather than just merging all sets M into one large unstructured sample $\mathcal{M}$. This algorithm is simple but has the potential disadvantage that, for large patterns, it may spend all its resources finding all the embeddings for the first sampled vertex.

Similar strategies have previously been proposed, such as in the work on triangle counting (e.g., Jowhari and Ghodsi 2005).

3.6 The Fürer-Kasiwiswanathan Approach

The algorithms we introduced in the previous section represent the baseline algorithms for finding the pattern embeddings. However, they can be very costly for larger patterns and domain graphs. Next, we propose an extension of

Algorithm 3 Random vertex sampling approach.

```
1: function RndV1Emb(graph $D$, pattern $P$, ordering $\mathcal{O}$)
2:     $\mathcal{M} \leftarrow \{\}$
3:     $R \leftarrow \{u \in V(D) \mid \lambda(u) = \lambda(\mathcal{O}_1)\}$ ▷ Find images for the first vertex of $P$
4:     while not timeout do
5:         Select a random $u \in R$
6:         $M \leftarrow$ RndV1Emb($D, P, \mathcal{O}, \{(\mathcal{O}_1, u)\}$)
7:         $\mathcal{M} \leftarrow \mathcal{M} \cup \{M\}$
8:     return $\mathcal{M}$

9: function RndV1Emb($D$,$P$,$\mathcal{O}$,$\varphi$) ▷ Find extensions for the partial embedding
10:     if timeout then
11:         return $\{\}$
12:     else if $|\varphi| = |V(P)|$ then
13:         return $\{\varphi\}$
14:     else
15:         $M \leftarrow \{\}$
16:         for all $\varphi' \in$ PartialExt$(D, P, \varphi, \{\mathcal{O}_{|\varphi|+1}\})$ do
17:             $M' \leftarrow$ RndV1Emb$(D, P, \mathcal{O}, \varphi')$
18:             $M = M \cup M'$
19:         return $M$
```

a theoretical sampling idea suggested by Fürer and Kasiviswanathan (2008) for unlabeled undirected graphs.

Figure 3.6 depicts one iteration of the the Fürer and Kasiviswanathan algorithm. Each iteration of the algorithm returns a single embedding and an estimate for the pattern's total number of embeddings. In a single iteration, it sequentially goes through all of the OBD's partitions. For each partition, it finds all possible extensions of the current partial embedding by matching the vertices in the partition. It then randomly samples one of these extensions and multiplies its estimate of the pattern's number of embeddings by the number of possible extensions found for the current partition.

The crucial part when constructing an OBD is to ensure that each partition is small. Since all vertices in a partition must be simultaneously matched, it is easier to a find a legal extensions when fewer vertices much be matched. If the decomposition provided as input is not an ordered bipartite decomposition, the sampling algorithm still converges to the correct value, but the convergence speed guaranteed by Fürer and Kasiviswanathan (2008) does not hold anymore.

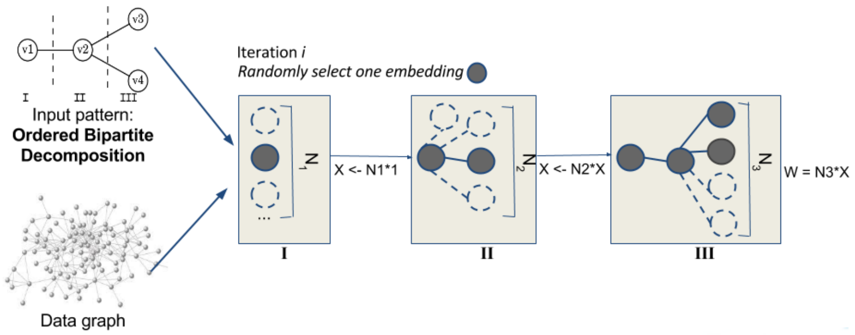

Figure 3.6: An illustration of one iteration of the Fürer and Kasiviswanathan approach.

Algorithm 4 Fürer and Kasiviswanathan approach.

```
1:  function FK-$\mathcal{V}$(domain graph $D$, pattern subgraph $P$, decomposition $\mathcal{V}$)
2:      $\mathcal{M} \leftarrow \{\}$
3:      while not timeout do
4:          $M \leftarrow$ FK-$\mathcal{V}(1, D, P, \mathcal{V}, 1, \{\})$
5:          if $M \neq \emptyset$ then
6:              $\mathcal{M} \leftarrow \mathcal{M} \cup \{M\}$
7:      return $\mathcal{M}$
8:
9:  function FK-$\mathcal{V}(W, D,P,\mathcal{V},i,\varphi)$
10:     if $i > |\mathcal{V}|$ then
11:         return $(\{(\varphi, W)\})$                         ▷ If all partitions embedded
12:     else if timeout then
13:         return $\{\}$
14:     $C \leftarrow$ PARTIALEXT$(D, P, \varphi, \mathcal{V}_i)$      ▷ Find all extensions for the partition
15:     if $C = \{\}$ then
16:         return $\{\}$
17:     Select $\varphi'$ randomly from $C$                 ▷ Randomly select one extensions
18:     return FK-$\mathcal{V}(W|C|, D, P, \mathcal{V}, i+1, \varphi')$
```

Algorithm 4 shows our extension of the Fürer and Kasiviswanathan approach to labeled graphs. The algorithm consists of two parts: the control loop (lines 2-7), and the single attempt procedure (lines 9-18). It receives the same parameters as Algorithm 3, except that it receives a decomposition $\mathcal{V}$ instead of an ordering $\mathcal{O}$. While the time limit is not exceeded, the control loop randomly samples, with replacement, a path in the search space that possibly leads to an embedding. The function FK-$\mathcal{V}$ returns a set of pairs (φ, W) where φ is the embedding found and W is a positive number equal to the inverse probability that an attempt will find that particular embedding. The expected value of this W (assuming implicitly 0 for the unsuccessful attempts) is the total number of embeddings.

A recursion step samples a set of images for a partition $\mathcal{V}_i$, extends the partial embedding accordingly, and calls the next recursion level. While doing so, it also computes the size $|C|$ of the set of options of which it has sampled only one. The product of these values $|C|$ is passed in the parameter W to the next recursion (becoming 0 when the attempt fails).

Example 23. *For example, if $\mathcal{V}_i = \{w1, w2, w3\}$, each vertex $w \in \mathcal{V}_i$ has a set of candidate matches C_w in D. Suppose these are $C_{w1} = \{a1, a2\}$, $C_{w2} = \{a3\}$ and $C_{w3} = \{a1, a4, a5\}$. Then each element of the Cartesian product $C_{w1} \times C_{w2} \times C_{w3}$ is a valid match for the vertices in $\mathcal{V}_i$ if the matching operator is homomorphism. In this case, $|C|$ would equal 6. In the case of subgraph isomorphism, embeddings mapping several vertices to the same image (here $\{(w1, a1), (w2, a3), (w3, a1)\}$) must be eliminated, so $|C|$ would equal 5.*

3.7 Finding Ordered Bipartite Decompositions

Recall that Algorithm 4 relies on having an ordered bipartite decomposition (OBD) of the pattern P for which it is estimating the number of embeddings. For efficiency, the size of each partition class in an OBD is important. Having more vertices in a single partition requires matching multiple vertices simultaneously, which is more computationally expensive. In particular, the size of set C in line 14 of Algorithm 4 may be exponential in $\mathcal{V}_i$. This implies that we want to minimize the number of vertices in each partition class. This also suggests that we prefer having OBDs that are as large as possible (i.e., have as many partition classes). Fürer and Kasiviswanathan (2008) assume that a pattern's OBD is given and do not address the problem of automatically finding an OBD for a given pattern. Thus, we propose two different search strategies for automatically finding an OBD.

3.7.1 Exact Search

Algorithm 5 presents a level-wise search for finding the largest possible OBD. First, it checks if P contains a triangle, because then no OBD exists. The maximal OBD size for pattern P is $|V(P)|$, which is the number of vertices in P. Starting with the largest possible size $s = |V(P)|$, the algorithm performs a search through the space of possible OBDs. In each iteration, it checks whether an OBD of size s exists by generating all possible candidate OBDs of size s. This entails enumerating all ways of dividing $V(P)$ into s partition classes, and then building one candidate for each possible ordering of the partition classes. It then applies the VALID_OBD function to check whether each candidate is a valid OBD (i.e., it satisfies the requirements of Definition 8). As soon as it finds a valid OBD, it is returned and the search terminates. If no OBD of size s is found, then s is decremented, and the search proceeds. If an OBD is found, its size is maximized because all larger decompositions were evaluated and none were valid OBDs.

Algorithm 5 Level-wise exact OBD search

```
1: function FindLargestOBD(graph $P$)
   return Largest ODB for $P$ or $\emptyset$ if no OBD exists
2:     if contains_triangle($P$) then
3:         return $\emptyset$
4:     $s \leftarrow |V(P)|$
5:     while $s > 0$ do
6:         for all ordered partitions $\mathcal{V}$ of $V(P)$ of size $s$ do
7:             if Valid_OBD($\mathcal{V}$) then
8:                 return $\mathcal{V}$
9:         $s \leftarrow s - 1$
10:    return $\emptyset$
```

3.7.2 Greedy Search

Algorithm 6 presents a depth-first search for finding an ODB for a pattern P. It greedily searches the most promising path and returns the first OBD found. Because it performs backtracking, it is guaranteed to find an OBD if one exists. However, its greedy nature means that it is not guaranteed to find the largest valid OBD.

The main algorithm loops through each node v in P, and builds a partial OBD $\mathcal{V} = (\{v\})$ consisting of a single partition class containing v (if there exists an OBD, then there exists one with a singleton as first partition class). Next, it calls GreedyOBD which recursively extends the partial OBD by adding one partition class at a time.

If GreedyOBD receives a valid and complete OBD (i.e., it contains all the nodes in P), it terminates and returns the current OBD. Otherwise, it generates a set of candidate extensions of $\mathcal{V}$. To do so, it first builds a graph $G' = (R, L)$, which has one node for each vertex in P that has not been assigned a partition class in the partial OBD $\mathcal{V}$. It has one edge $\{x, y\} \in L$ for each $x, y \in R$ such that x and y share a neighbor in P that already appears in $\mathcal{V}$, which implies that x and y must be in the same partition class. Because a node $x \in R$ may neighbor multiple nodes in $\mathcal{V}$, the set of candidate extensions $\mathcal{C}$ consists of each connected components in G'. Next, line 15 checks if any candidate partition requires assigning the same label to neighboring vertices in P, which violates the definition of an OBD, meaning no complete OBD can be constructed by extending this partial ODB. In this case, the algorithm backtracks. Otherwise, it creates a set of candidates extensions by concatenating each element (i.e., set of vertices) in $\mathcal{C}$ as a partition class to the end of $\mathcal{V}$. It sorts the set of candidates from smallest to largest on the basis of the size of the newest partition class. This heuristic is an attempt to favor OBDs with small partition classes. It then recursively calls GreedyOBD on each candidate in this order.

3.7.3 Empirical Evaluation of OBD Search

We empirically compare the run time performance of the two OBD search algorithms. We generate all simple connected graphs with between five and eight vertices and use both Algorithm 5 and 6 to find an OBD for each generated pattern. Table 3.1 reports the mean, median and maximum run time in seconds of both algorithms on each pattern size. While the greedy approach does not guarantee finding an optimal solution, on average it is several orders of magnitude faster than the level-wise approach. As it scales better than the exact approach, our empirical evaluation in Section 3.9 employs the greedy approach to find an OBD for each evaluated pattern.

Algorithm 6 Depth-first greedy OBD search.

```
1: function GreedyFindOBD(graph $P$)    ▷ return OBD for $P$ or $\emptyset$ if no OBD exists

2:     for all vertices $v \in V(P)$ do
3:         $result = $ greedyOBD$((\{v\}), P)$
4:         if $result \neq \emptyset$ then
5:             return $result$
6:     return $\emptyset$

7: function greedyOBD(partial OBD $\mathcal{V}$, graph $P$)
8:                                   ▷ $\mathcal{V}$ contains all nodes of $P$ and is a valid OBD
9:     if Complete$(\mathcal{V})$ && Valid_OBD$(\mathcal{V})$ then
10:        return $\mathcal{V}$
11:    else
12:        $R \leftarrow V(P) \setminus \cup_{S \in \mathcal{V}} S$        ▷ Vertices not yet in partition class
13:        $L \leftarrow \{\{x, y\} \in R \mid \exists V \in \mathcal{V} : N_P(x) \cap V \neq \emptyset \wedge N_P(y) \cap V \neq \emptyset\}$
14:        $\mathcal{C} \leftarrow$ ConnectedCompents$((R, L))$)
15:        if $\exists C \in \mathcal{C}, \exists x, y \in V(C) : \{x, y\} \in E(P)$ then
                                   ▷ $x$ and $y$ must be apart $\Rightarrow$ no OBD possible
16:            return $\emptyset$
17:        for all $C \in \mathcal{C}$ do
18:            $result \leftarrow$ greedyOBD$(\mathcal{V} \cdot V(C), P)$        ▷ Concatenate $C$ to $\mathcal{V}$
19:            if $result \neq \emptyset$ then
20:                return $result$
21:    return $\emptyset$
```

	Exact			Greedy		
|V|	Mean	Median	Max	Mean	Median	Max
5	0.0014	<0.0001	0.0096	0.0002	0.0003	0.0004
6	0.0106	<0.0001	0.1611	0.0004	0.0004	0.0010
7	0.0618	<0.0001	2.8392	0.0005	0.0005	0.0025
8	0.3894	<0.0001	54.9054	0.0008	0.0007	0.0084

Table 3.1: This table reports the mean, median and maximum run times in seconds for the exact and depth-first search algorithms for finding an OBD on all simple connected graphs with between five and eight vertices.

3.8 Computing Statistics

The algorithms described in the previous sections all generate a sample of embeddings. In this section we will discuss how we can compute a number of statistics from such samples, and we will study their accuracy.

3.8.1 Common Statistics

Let P be a pattern and D a domain graph. We first define the statistics we will consider.

Count. The *count* statistic just returns $|\mathcal{V}al(P, D)|$, the number of embeddings of P in D.

Mean and variance Let $t : \mathcal{R}ange(P, D) \to \mathbb{R}$ be a function mapping values of embeddings on real numbers. Then, $\mu_t(X) = \sum_{x \in X} t(x)/|X|$ and $var_t(X) = \sum_{x \in X}(t(x) - \mu_t(X))^2/|X|$.

Confusion matrix. Some vertices of the pattern could be labeled by finite sets. In that case, it may be interesting to compare the relative frequencies with which these vertices map on vertices with certain labels.

More formally, let $S \subseteq V(P)$ be a set of vertices of P. For $x \in \mathcal{R}ange(S : P, D)$, a set of possible values of the vertices in S, and $y \in \mathcal{V}al(P, D)$, the value of a particular embedding, we define $I_{S \to x}(y) = 1$ if $y|_S = x$ and $I_{S \to x}(y) = 0$ if $y|_S \neq x$. Next, for a set of values $\mathcal{V}al(P, D)$, $freq_{S \to x}(\mathcal{V}al(P, D)) = \mu_{I_{S \to x}}$ and hence $freq_{S \to x}(\mathcal{V}al(P, D))$ is the fraction of embeddings of P in D for which the vertices in S get the value x. We define $freq_S(\mathcal{V}al(P, D)) = \{(x, freq_{S \to x}(\mathcal{V}al(P, D)) \mid x \in \mathcal{R}ange(S : P, D)\}$. Note that, $freq_S(\mathcal{V}al(P, D))$ is a confusion matrix (or tensor), as for each possible combination of values for the vertices in S it gives the observed (relative) frequency.

3.8.2 Estimators

We now discuss how to compute statistics from samples. In the above, we defined the elements of the confusion matrix statistic using a mean statistic, so we can limit our discussion to *count*, μ and *var* here. In the empirical evaluation section, we will focus more on the confusion matrix statistic.

Let us first consider the random sampling. It returns a set $\mathcal{M} = \{M_i\}_{i=1}^n$ of groups of embeddings with n the number of main loop iterations of the algorithm. Each group $M_i = \{\varphi_{i,j}\}_{j=1}^{n_i}$ of embeddings is generated together during one iteration of the random approach and shares the same image for the first pattern vertex. We can estimate $count(P, D)$ by $count^{rnd}(P, D) = |V(P)| \sum_{i=1}^{n} n_i/n$. We can estimate $\mu_t(P, D)$ by $\hat{\mu}_t^{rnd} = \sum_{i=1}^{n} \sum_{j=1}^{n_i} t(val_D(\varphi_{i,j}))/ \sum_{i=1}^{n} n_i$. This is a consistent estimator (it converges to $\mu_t(P, D)$ when n gets large) but even though $count^{rnd}$ and $\sum_{i=1}^{n} \sum_{j=1}^{n_i} t(val_D(\varphi_{i,j}))$ are unbiased estimators of the count and the sum, $\hat{\mu}_t^{rnd}$ is not an unbiased estimator itself. For instance, suppose that the first vertex has two possible images u_1 and u_2, u_1 is in three embeddings with t-values 1, 2 and 3 and u_2 is in one embedding with t-value 6. Then, a single iteration of the sampling algorithm will either produce values 1, 2 and 3 leading to an estimation $(1 + 2 + 3)/3 = 2$ or a single value 6, leading to an estimate of 6. On average, we expect an estimation of 4, while the average of all values is 3.

Next, we consider the FK-$\mathcal{V}$ algorithm. It returns a set $\mathcal{M} = \{(\varphi_i, w_i)\}_{i=1}^{n}$ of independently sampled weighted embeddings. Fürer and Kasiviswanathan (2008) showed that $count^{fk}(P, D) = \sum_{i=1}^{n} w_i/n$ is an unbiased estimator. For μ_t we can derive an unbiased estimator too (but omit details due to lack of space). Let $W = \sum_{i=1}^{n} w_i$ and $W_2 = \sum_{i=1}^{n} w_i^2$. Then, $\hat{\mu}_t^{fk}(P, D) = \sum_{i=1}^{n} w_i t(val_D(\varphi_i))/W$ is an unbiased estimator of $\mu_t(P, D)$. Moreover, we can estimate the square error from the sample with $\mathbb{E}[\hat{\mu}_t^{fk}(P, D) - \mu_t(P, D) = (W_2/W.(W^2 - W_2))$ $\sum_{i=1}^{n} w_i(t(val_D(\varphi_i)) - \hat{\mu}_t^{fk})^2$. It is straightforward to show that these estimators remain unbiased even if Fürer-Kasiwiswanathan is run with a decomposition which is not an OBD. The main difference is that in that case the error converges more slowly to zero as the sample size increases.

3.9 Experiments

We empirically compare the performance of the following four approaches:

Exhaustive: This is the approach of Algorithm 2.

Random: This is the approach of Algorithm 3 that randomly samples an initial vertex in the domain graph from which to start the search for the embeddings of a pattern.

FK-OBD: The extended Fürer and Kasiviswanathan approach outlined in Algorithm 4 that uses greedy search to find an OBD for a pattern. It only applies to generated patterns that have an OBD.

FK-AD: The extended Fürer and Kasiviswanathan approach outlined in Algorithm 4 that works with an arbitrary decomposition (AD) of a pattern.

To construct an arbitrary decomposition, we simply flatten the OBD, which means that each partition contains exactly one node.

The goal of the evaluation is to address these three questions:

1. How well do the sampling algorithms approximate the true embedding statistics obtained using the exhaustive approach?
2. What is the relative runtime of the algorithms?
3. How does having an OBD vs. an arbitrary decomposition (AD) affect the performance of the Fürer-Kasiwiswanathan algorithm?

Next, we describe our datasets, the experimental methodology, and then we present and discuss the results.

3.9.1 Datasets

We evaluate the algorithms using 12 synthetic datasets and two real-world datasets. Our real-world datasets are initially stored in a relational format. We use a standard transformation (e.g., (Richards and Mooney 1992)) to convert these datasets into a graph form. The general idea is that graphs consist of constants connected through their relations.

Synthetic Datasets

We created a series of synthetic datasets that contain the following node types: *person, salary, satisfaction, man, woman, married* and *friends*. We create two types of graphs by varying the way friend nodes are connected within the graph. The first employs the Erdős-Rény random graph model, and adds them randomly. The second adds the nodes according to the power law distribution of the Barabási-Albert algorithm (Barabasi and Albert 1999). Finally, each person receives a salary and a satisfaction, 40% of the people are male, the rest are female, and 70% of the people are married. Table 3.2 shows the synthetic graphs and their characteristics.

			Power Law		Random	
#Nodes	#Edges	#*Friends*	Name	MaxDeg	Name	MaxDeg
9335	13670	4985	S1	131	R1	24
93485	136970	49985	S2	383	R2	32
14295	23590	9945	S3	174	R3	36
143445	236890	99945	S4	593	R4	44
24140	43280	19790	S5	225	R5	60
243290	436580	199790	S6	736	R6	69

Table 3.2: Characteristics of the synthetic graphs. MaxDeg denotes the maximum degree of a data graph.

Real-world Datasets

For both real-world datasets we transformed directed or labeled edges into 2-paths with an intermediate labeled vertex such that our graphs only have vertex labels. Next, we describe the datasets in more detail.

DBLP dataset We use the DBLP[1] bibliography database from the ArnetMiner repository (Tang et al. 2008). We only consider data from 2001 and 2002, to ensure that the exhaustive approach is computationally feasible. The DBLP domain graph consists of *paper, coauthored, reference* and *citations* nodes. One *paper* node is added for each paper in the database. For each pair of papers that share an author, a *coauthored* node is added to the graph and connected to those papers. For each pair of papers where one paper refers to the other, a *reference* node is added to the graph (using an intermediate vertex *dir* to designate the direction) and connected to those two papers. Finally, each paper is connected to a *citations* node, which was discretized into three categories: *high*, *med* and *low*. The graph contains 393,230 vertices, 447,650 edges, and has a maximum degree of 1,036.

YEAST dataset The Yeast Protein data comes from the MIPS (Munich Information Center for Protein Sequence) Comprehensive Yeast Genome Database (Mewes et al. 2000). We use the version of the dataset (Davis and Domingos 2009) from the Alchemy repository, with the modification that we removed all self-referential links.[2] The graph has the following node types: *protein, location, function, phenotype, class, enzymes,* and *interaction*. A node is

[1]http://www.informatik.uni-trier.de/~ley/db/index.html
[2]http://alchemy.cs.washington.edu

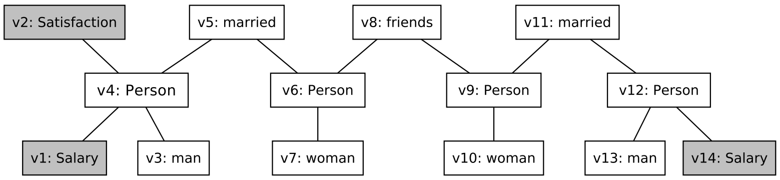

Figure 3.7: Pattern graph $P_{sat-sal}$ for our example. Shaded nodes are the target nodes of the pattern.

added to the graph for each protein. For each pair of proteins that interact, an *interaction* node is added to the graph and connected to those proteins. For each protein, a node is added for all of its locations, functions, phenotypes, classes and enzymes. The graph contains 16,233 vertices, 18,355 edges, and has a maximum degree of 124.

3.9.2 Experimental Setup

Next, we introduce our experimental methodology which for the real-world datasets consists of pattern generation, sampling of interesting patterns based on a criterion and evaluation of the approaches.

Generating Pattern Graphs

We now describe our method for generating pattern graphs for the synthetic and the real-world data.

Synthetic pattern graph For the synthetic dataset we only consider one pattern graph, which is depicted in Figure 3.7. This pattern is a slight modification of a phenomena described in (Ariely 2008).

Generating pattern graphs for real-world data We want to evaluate our approach on a variety of patterns. However, even with a restriction on the pattern size, it is computationally too demanding to evaluate all possible candidate patterns. Hence, we generate patterns of increasing size in a level-wise manner, and evaluate a subset of patterns of each size.

Each pattern is an undirected graph. We vary the pattern size from 4 to 15 nodes. Given a set C_L of patterns of size L, we generate a candidate set C_{L+1} of patterns of size $L+1$ by enumerating all valid extensions of each pattern in C_L. Each pattern is extended by first adding a new node to it. We consider adding all node types. Then, new candidates are created for all ways of adding up to $m = n \cdot (n-1)/2$ edges between the new node and n existing nodes in the pattern. We remove duplicates from the candidate set by checking for isomorphism between pairs of candidate patterns.

We focus our evaluation on a sample of patterns where each selected pattern is neither too frequent nor too infrequent as these tend to be more interesting patterns for analysis. Thus, we randomly sample patterns from C_{L+1} and run FK-OBD or FK-AD for one hour. The former is used if a pattern has an OBD, and the latter if it does not. We continue sampling until we find 100 patterns that meet the following criterion:

$$\sqrt{|V(D)|} - 3 \cdot N_{std} \leq N_{avg} \leq |V(D)| + 3 \cdot N_{std},$$

where $|V(D)|$ denotes the number of nodes in the data graph D, N_{avg} denotes the average and N_{std} the standard deviation of the estimated number of embeddings after the one-hour run.

Pattern Evaluation

We evaluate all algorithms on the selected patterns for each size. We run each algorithm for a maximum of K time units and record intermediate results after each T time units. For DBLP, we set K to ten hours and T to five minutes. For YEAST and the synthetic datasets, K is ten minutes and T is five seconds. This means that we record 120 intermediate results for each pattern, regardless of the dataset or method.

3.9.3 Evaluation Tasks and Metrics

We consider two different evaluation tasks.

Frequency estimation This corresponds to the standard pattern mining task, where the support of a pattern in the data is calculated. For frequency estimation of a pattern, we report the relative error of a pattern's estimated frequency (using a sampling algorithm) up until time t, denoted as $\#\hat{emb}_i(t)$, versus its true frequency as computed by the exact approach. The relative error is given by the following formula:

$$RelError_i(t) = \frac{|\#emb_i - \#\hat{emb}_i(t)|}{\#emb_i} \tag{3.1}$$

Parameter estimation for LBNs In the introduction to this thesis and this chapter we mentioned that estimating pattern frequency is also a subtask in parameter learning for statistical relational learning formalisms such as Markov logic networks (MLNs) (Richardson and Domingos 2006), Bayesian logic programs (BLPs) (Kersting, De Raedt, and Kramer 2000), and logical Bayesian networks (LBNs) (Fierens, Blockeel, Ramon, et al. 2004), amongst others. We will focus on parameter estimation for LBNs.

Section 3.3 explained how the dependencies in relational formalisms can be expressed as pattern graphs. We now briefly describe how these dependencies are represented in LBNs. For a more detailed discussion on LBNs please consult (Fierens, Blockeel, Ramon, et al. 2004). Parameter estimation in LBNs requires estimated conditional probability distributions (CPD) of the form $P(X \mid \mathbf{Y} \leftarrow \texttt{Context})$. In these CPDs, X is a single random variable that can take on multiple values, $\mathbf{Y}$ is a set of random variables and `Context` defines a set of conditions that specifies when the CPD is valid (e.g., there only exists a random variable for a student's grade in a course `C` if she has taken course `C`.).[3] The CPD captures the conditional probability of X for each joint assignment of values to $\mathbf{Y}$, when `Context` holds in the data. Thus, in a constructed pattern we must specify which nodes represent each of X, $\mathbf{Y}$, and `Context`. For example, in Figure 3.7 shaded nodes represent X, $\mathbf{Y}$ (target) nodes, while other nodes represent `Context`. When creating the patterns for the DBLP dataset, X, $\mathbf{Y}$ are nodes labeled with *citations*. Nodes labeled as $references, dir, coauthored, citations = low, citations = med, citations = high$ represent `Context`. For YEAST based patterns X, $\mathbf{Y}$ can be nodes labeled with $function, location, protein\_class, enzyme, phenotype$. Each of these has a range of values. Nodes labeled with *interaction* and $predicate = value$ where $predicate \in \{function, location, protein\_class, enzyme, phenotype\}$ and $value \in range(predicate)$ make `Context`.

[3]Note that this is a slight simplification as LBN uses first-order logic to perform parameter tying across multiple random variables.

To make the results less susceptible to fluctuations arising from infrequently occurring combinations, we employ the standard Laplace smoothing of our estimates. Furthermore, we aggregate each value combination that has a relative frequency of $< 1\%$ in the exhaustive approach into a single "default" row. In probabilistic models, this is known as using a default table representation for the local probability distributions (Friedman and Goldszmidt 1996).

In order to compare the quality of the estimated and the true distribution for a pattern we calculate the Kullback-Leibler divergence (KLD).

$$KLD(P||Q) = \sum_i p(i) \times \log_2 \left(\frac{p(i)}{q(i)} \right) \tag{3.2}$$

The estimated distribution (P) is obtained by one of the sampling algorithms, and the true distribution (Q) is obtained by the exhaustive algorithm. In order to summarize results over a large number of patterns of a specific size, we report the average KLD (Avg_KLD).

3.9.4 Results

Next, we present the results on the synthetic and the real-world datasets.

Synthetic datasets

The goal of the synthetic experiments is to compare the algorithm's performance on random and non-random graphs. In these experiments, the exhaustive approach always finished within the time limit. Figure 3.8 and Figure 3.9 show the performance of each algorithm on average KLD and average relative error, respectively. On average, FK-OBD has a better KLD and smaller relative errors than Random and FK-AD. Finally, the performance of FK-OBD and FK-AD tends to be better than random sampling for power law graphs. This provides some evidence that FK-OBD and FK-AD will still perform well even on non-random graphs where the theoretical analysis does not hold.

DBLP and YEAST

Table 3.3 presents the runtime of the exhaustive approach and the percentage of exhaustive runs that failed to finish within our time limit. On the DBLP dataset,

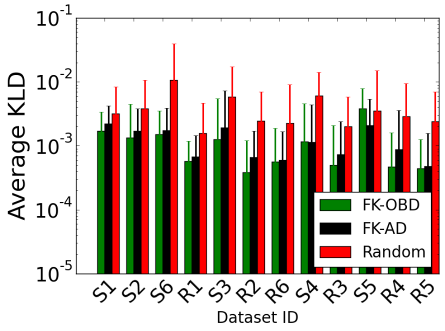

Figure 3.8: KLD averaged over all sampling iterations as a function of the synthetic datasets. Datasets are sorted by the increasing number of true embeddings from left to right. Error bars represent the standard deviations.

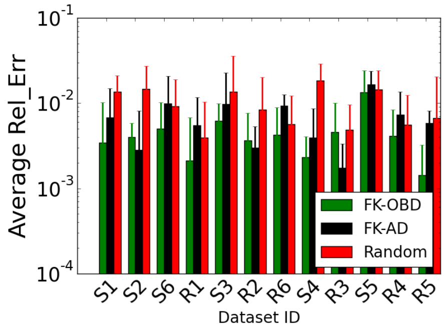

Figure 3.9: Relative error averaged over all iterations of the sampling algorithms as a function of the synthetic datasets that are sorted by the increasing number of true embeddings. Error bars represent the standard deviations.

the exhaustive algorithm starts to exceed the time limit for patterns of size ≥ 8 whereas on YEAST this happens for patterns of size ≥ 10. This is because patterns that increase the runtime (e.g., those that contain cycles) require more nodes due to the more sparsely connected graph. As expected, the runtime of the exhaustive approach tends to increase as the pattern size increases. The trend of increasing variance with the increase of the pattern size indicates that the selected patterns fail on two extremes, with exhaustive finishing rather fast or needing most of the time limit to finish.

We analyzed what is the percentage of sampled patterns having an OBD. For DBLP related patterns 38.2% of patterns we sampled had no OBD, but only 0.5% out of these were selected based on our criterion. For the YEAST dataset, all the patterns we sampled had an OBD. Actually, for YEAST we did not expect any triangles in the pattern that would cause an OBD to not exist. The only way a triangle could appear in patterns made for YEAST is to have a direct *protein* to *protein* connection. However, protein nodes can only be connected through an intermediate *interaction* node. In Figure 3.10 we show the relative error as a function of time for applying FK-$\mathcal{V}$ on patterns that do not have an OBD (FK-No-OBD) opposed to patterns that had an OBD (FK-OBD). It is evident that the performance of FK-OBD converges even when it receives a pattern that does not have an OBD.

	DBLP			YEAST		
		Runtime(min.)			Runtime(min.)	
Pattern size	%exh. time-out	Mean	Stdev	%exh. time-out	Mean	Stdev
4	0.0	0.66	0.95	0.0	0.06	0.05
5	0.0	14.44	52.57	0.0	0.08	0.06
6	0.0	33.23	95.15	0.0	0.17	0.09
7	0.0	37.04	111.26	0.0	0.09	0.09
8	3.0	63.44	149.91	0.0	0.17	0.13
9	1.0	56.15	127.01	0.0	0.36	0.85
10	10.0	100.65	190.31	0.0	0.42	0.57
11	12.0	128.21	204.52	3.0	1.36	2.45
12	13.0	128.64	204.87	1.0	0.89	1.60
13	20.0	167.17	233.30	3.0	2.04	2.51
14	19.0	154.41	229.63	12.0	3.23	3.52
15	8.6	117.59	191.51	27.0	4.45	3.85

Table 3.3: Runtime of exhaustive approach (in minutes) and percentage of timeout exhaustive runs per pattern size for the DBLP and YEAST dataset.

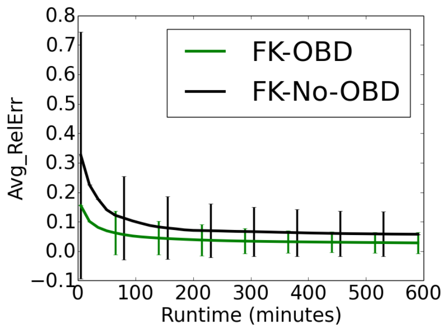

Figure 3.10: DBLP: Relative error as a function of time averaged over patterns that do not have an OBD. Error bars represent standard deviations.

Avg_KLD and Avg_RelErr In the following plots, we omit those patterns where exhaustive failed to finish within the time limit as we lack the ground truth answers for them. Figure 3.11 presents results on the average KLD for both datasets. Figure 3.12 presents results on the average relative error for both

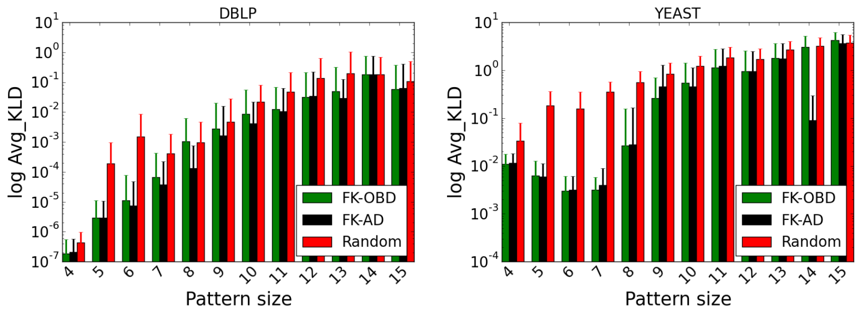

Figure 3.11: Average KLD represented in logarithmic scale as a function of the pattern size. The average is taken over all the patterns per pattern size and the error bars represent standard deviations. Results are shown only for those patterns for which exhaustive finished within 10 hours for DBLP, and 10 minutes for YEAST.

datasets. The results show that the YEAST dataset is slightly more problematic for the random vertex approach than DBLP. We can say that FK-OBD and FK-AD have similar performance trends and are on average better than the random vertex approach on both metrics.

Runtime performance Ultimately, one goal of sampling is to achieve similar performance but at a fraction of the runtime of the exhaustive approach. We illustrate this by allowing each sampling approach to run for a percentage of the total runtime of the exhaustive approach. For example, if the exhaustive approach finished in 5 hours, then 10% represents FK-OBD and FK-AD estimated results after 30 minutes. Moreover, we focus on more challenging patterns by omitting any pattern where FK-AD had less than 10% relative error on the estimated number of embeddings in the first interval (5 minutes for DBLP and 5 seconds for YEAST). The results for the DBLP dataset are shown in Figure 3.13, and for YEAST in Figure 3.14. As we can see the performance of FK-OBD and FK-AD is rather similar. The fact that for smaller pattern sizes the error bars do not decrease (or stay almost constant) is caused by the fact that for these levels exhaustive on average finished its execution rather early. This means that sampling algorithms are given less time for sampling.

Exhaustive runs with timeout To show the performance on the patterns where exhaustive exceeded the time limit, we show how each approach converges to its final estimate by assuming these estimates are the ground truth. The results

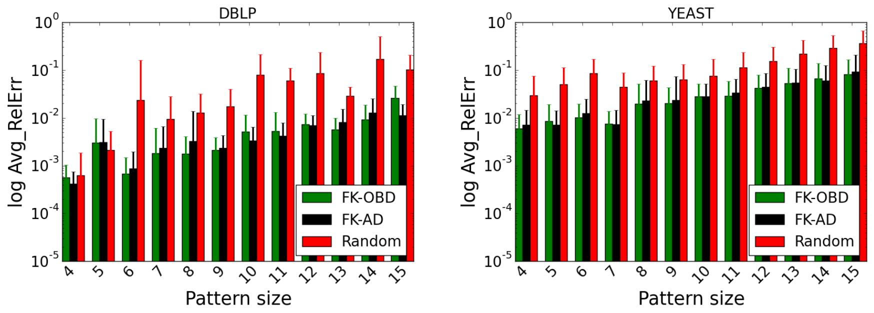

Figure 3.12: Average relative error represented in logarithmic scale as a function of the pattern size. The average is taken over all the patterns per pattern size and the error bars represent standard deviations. Results are shown only for those patterns for which exhaustive finished within 10 hours for DBLP, and 10 minutes for YEAST.

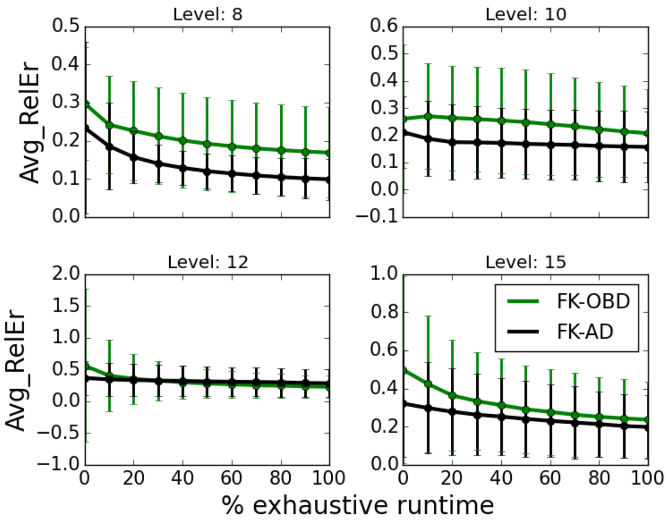

Figure 3.13: DBLP: Average relative error plots for a subset of results: pattern size 8, 10, 12 and 15. The patterns used are those for which exhaustive finished in the given time frame and those for which FK-AD had a relative error on the number of embeddings larger than 10% in the first interval of sampling (i.e., 5 minutes for DBLP and 5 seconds for YEAST). Error bars represent standard deviations after each 10% execution.

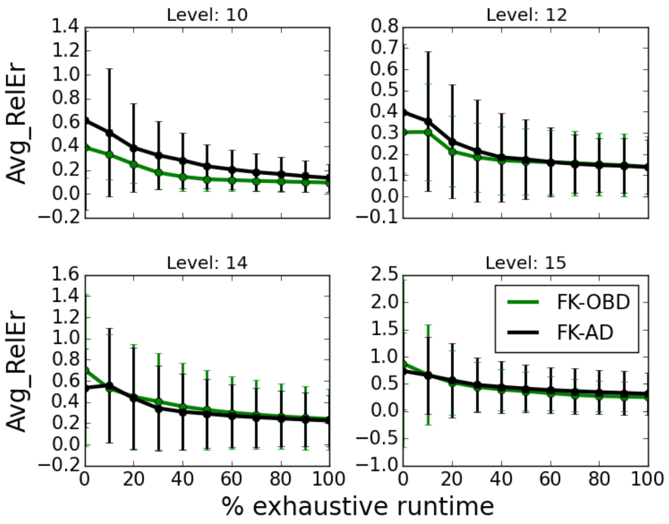

Figure 3.14: YEAST: Average relative error plots for each pattern size. The patterns used are those for which exhaustive finished in the given time frame and those for which FK-AD had relative error on number of embeddings larger than 10% in the first interval of sampling (i.e., 5 minutes for DBLP and 5 seconds for YEAST). We only show results for pattern sizes 10, 12, 14 and 15. Error bars represent standard deviations after each 10% of the execution.

for DBLP and YEAST are shown in Figure 3.15. The best convergence for all the approaches occurs when FK-OBD and FK-AD are taken as the ground truth. This is especially the case for the DBLP dataset for which random approach has a bad convergence trend in the beginning of the sampling. This happened because the random approach had to visit many vertices to start finding embeddings for some specific patterns. Given that we already witnessed the convergence of FK-OBD and FK-AD in the results presented before, we can say that these patterns are the ones that would make exhaustive run for quite a long time.

3.9.5 Discussion

Finally, we address the experimental questions posed at the beginning of this section in light of the presented results. The results on the real-world datasets show that both FK-OBD and FK-AD converge rather early in the sampling. The

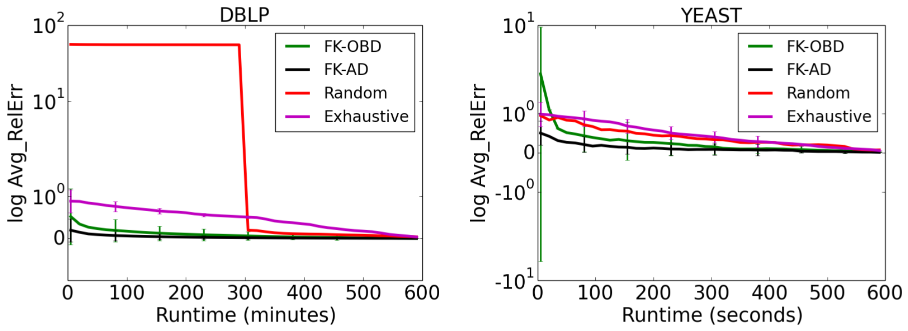

Figure 3.15: Average relative error for incomplete runs of the exhaustive approach for the DBLP and YEAST datasets demonstrating how each approach converges to its final estimate by assuming these estimates are the ground truth.

experiments also show that the sampling algorithms on average converge very fast to the estimates with a low error. Furthermore, these results highlight that the FK is a viable strategy for non-random graphs. The random vertex approach performed worse than the two FK approaches.

FK-OBD and FK-AD seem to perform similarly, thus even when the decomposition is not an OBD the approach seems to converge in practice. One possible reason is that because the size of AD is larger, it is faster to match. Hence, it makes more observations than FK-OBD which needs to calculate all possible combinations of values for nodes in each partition of an OBD. The larger the partition, the longer it takes to finish one iteration of sampling.

Finally, we examined the results for which the exhaustive approach failed to finish in a given time. We notice fast convergence trends of the approaches when we use FK-OBD and FK-AD as the ground truth.

3.10 Related Work

Data in domains, such as the Semantic Web, social networks, citation networks, biology, relational learning and geography among others, are often naturally represented as a graph. Therefore, analyzing large graphs or networks is a very active research field. Furthermore, problems related to the analysis (searching, counting, etc.) of subgraphs in large network graphs attracts significant attention. Next, we present some of the more prominent lines of research concerned with subgraph analysis and their relation to this paper.

Many of the following approaches employ subgraph identification, either by subgraph isomorphism or subgraph homomorphism as a subtask. Typically, they employ existing state-of-the-art algorithms for this task (e.g., Cordella et al. (2004) and Ullmann (1976)).

One line of work looks at defining features of graphs. Graph particles are a promising approach for *graph characterization and comparison* (Pržulj 2007; Shervashidze et al. 2009; Bordino et al. 2008). Graph particles are usually small subgraphs that contain up to five nodes, and are sometimes called graphlets. The distribution of graph particles can be used as a "fingerprint", a characteristic invariant, of a graph. Then, the similarity between pairs of graphs can be measured by comparing their distributions of graph particles. *Graph querying* (Di Natale et al. 2010; Giugno and Shasha 2002) focuses on searching for a graph in a database of graphs. Here, most research focuses on identifying graph features (e.g., paths, walks, subtrees, small subgraphs, etc.) for filtering and indexing in order to enable faster searching and querying of graphs.

Frequent pattern mining in graph databases is another active topic of research (Yan and Han 2002; Inokuchi et al. 2003). This usually entails finding frequent subgraphs, which are also called *motifs* in biological networks (Kashtan et al. 2004; Wernicke 2005). Motifs are subgraphs that appear more often than expected according to a null model. Although subgraph isomorphism is an integral sub-task in this problem, the main research focus is determining the significance of subgraphs, which is measured by their support in the data. Usually, these approaches only consider small subgraphs since the goal is to find all subgraphs that meet the support threshold. Some approaches (e.g., Baskerville et al. (2007)) use approximate methods to scale to larger subgraphs.

Finally, general graph sampling approaches explore how to obtain a *representative sample* of a graph for a specific purpose (Leskovec and Faloutsos 2006). Some approaches, like the work of Zou and Holder (2010), rely on sampling for performing subgraph analysis by first sampling from a single large graph and then mining the sample for frequent subgraphs. They empirically assess the viability of various graph sampling approaches for the frequent subgraph mining.

The two key differentiating factors of our work are that we focus on approximating the frequency of a subgraph in larger graph (i.e., the inverse task of Zou and Holder (2010)), and we consider relatively large subgraphs (up to 15 nodes). Perhaps the most closely related work is that of Bordino et al. (2008) who approximately count the subgraphs for graph characterization. But again, they only consider small subgraphs.

Within SRL, the early work of Kok and Domingos (2005) recognized the potential of sampling to improve scalability. They explored both sampling the facts that appear in the data and the number of true groundings of a clause. More recently, Venugopal et al. (2015) proposed an approach specific to Markov logic that improves efficiency by counting only satisfied groundings, which is equivalent to counting paths in a graph. Our work is similar to the use of graph databases to scale lifted inference and learning by performing approximate counting (Das et al. 2016).

3.11 Conclusions and Future Work

In this chapter we studied sampling properties of graph pattern embeddings. This task is important in many areas of graph data mining, including (approximate) pattern mining and statistical relational learning. We tested several different sampling algorithms, studied how to obtain statistics from the embeddings they generate, and performed an extensive empirical comparison.

We observed that a strategy based on the theoretical algorithm by Fürer and Kasiviswanathan (2008) performed well in complex cases and the convergence is observed in all the experiments. Moreover, the convergence seems to have happened early in the stages of sampling. In the theoretical work of Fürer-Kasiwiswanathan an ordered bipartite decomposition (OBD) of a pattern was assumed to be given as an input. In this work we presented a heuristic approach of calculating the OBD of reasonably large patterns. We showed that even for patterns that do not have an OBD, the Fürer-Kasiwiswanathan algorithm turned out to be rather robust to the use of non-ODB decompositions.

There are several possible directions for future work. First, statistical relational learning has more complex inference tasks and statistics-collection challenges next to the one presented here. Second, while related work focuses on rather simple patterns, we have presented an algorithm allowing for sampling the embeddings of reasonably complex patterns. This is a crucial step towards developing better pattern mining algorithms for large networks.

4

Hybrid Relational Dependency Networks

"Open communication channels, Lister. Broadcast on all known frequencies and in all known languages, including Welsh."
Rimmer, Red Dwarf

4.1 Introduction

This chapter tackles another challenge posed by real-world relational data: the attributes of objects can take on discrete or continuous values. Consequently, one needs a representation for which inference and structure learning can be performed in these hybrid domains.

To address the problem, this chapter presents hybrid relational dependency networks (HRDNs) which upgrade an existing SRL formalism, relational dependency networks (RDNs), to hybrid domains. Even though semantically RDNs represent an approximation to the joint probability distribution, we chose it for several reasons. First, there exists a rather simple and computationally efficient method for learning the structure and parameters of an RDN from data.

Second, unlike relational formalisms based on Bayesian networks, RDNs allow cyclic dependencies that are ubiquitous in relational domains (e.g., grades of students who are friends are correlated), and are exploited for performing collective inference. Third, RDNs are expressive in that they represent a collection of any regression or classification techniques that can be combined using the Gibbs sampling procedure to obtain an approximate joint distribution.

4.1.1 Contributions and Bibliographical Note

The main contributions of this chapter are two-fold. First, we describe the semantics of hybrid relational dependency networks (HRDNs). This formalism will pave the way for developing the challenging tasks in hybrid SRL such as structure learning. The second contribution is proposing a number of conditional probability models to use for quantifying the hybrid dependencies in HRDNs.

This chapter is based on the following published paper:

> *Irma Ravkic, Jan Ramon, Jesse Davis (2015) "Learning relational dependency networks in hybrid domains". In: Machine Learning Journal 2015, volume 100, pp. 217 - 254*

The author of the dissertation contributed to defining the representation and semantics of HRDNs based on first-order logic, and proposing a number of hybrid local distributions that are adequate for modelling hybrid domains.

4.1.2 Chapter Structure

The remainder of this chapter is structured as follows. We first introduce hybrid relational dependency networks with an example, and its formal syntax and semantics. Then we discuss several hybrid conditional probability models. We end the introduction to HRDNs by discussing inference. This is followed by related work and short overture to the task of learning the structure of HRDNs, which will be covered in Chapter 5. We end this chapter by providing the conclusions.

4.2 Hybrid Relational Dependency Networks

We now describe hybrid relational dependency networks (HRDNs), our proposed extension to RDNs for hybrid domains. First, we describe how to incorporate continuous variables. Second, we describe how to represent the CPDs. Third, we briefly describe how to perform inference in HRDNs.

4.2.1 Hybrid Relational Dependency Networks by Example

We will now discuss the Example 2.3 by focusing on its hybrid aspect.

Example 24. *There are six relational tables representing properties and relationships of students, courses and professors. Attributes such as* intelligence *and* numHours *have numeric ranges. For example, the maximum value for IQ can be* 228*, and the number of hours needed to prepare for a course might be at most* 180*. An interesting relational feature to query is the aggregate intelligence quotients of one's friends who take the same course. As this aggregation is done for a numeric attribute, an appropriate aggregation, such as averaging, needs to be performed. Also both of these variables can be Gaussian distributed. For example, IQ is roughly normally distributed and IQ tests are constructed to have a mean of* 100 *and a standard deviation of* 15*.*

In Section 2.4.2 we introduced relational dependency networks, an SRL formalism we will upgrade to hybrid domains in this dissertation. In order to accomplish that we need to modify the components and definitions of RDNs. These components are : a) random variable declarations, b) relational features, and c) local distributions.

4.2.2 Syntax of HRDNs

First, to introduce continuous variables, it suffices to declare the range of a predicate to be an interval of the real numbers. Each continuous randvar associated with such a predicate can then take on any value from this interval.

Example 25. *We could define a predicate* `numHours/1` *with the following random variable declaration:*

$$\texttt{random}(\texttt{numHours}(\texttt{C})) \leftarrow \texttt{course}(\texttt{C})$$

that represents the number of hours needed to study for a course C*. The range of this predicate can be the following interval:*

$$\texttt{range}(\texttt{numHours}(\texttt{C})) = [20.0, 180.0]$$

Second, we need to modify the definition of a relational feature to account for the fact that both atoms and aggregation functions can have continuous ranges.

Definition 9. (Numeric Relational Feature) *A* numeric relational feature *has the same form as a discrete relational feature,* $\texttt{agg}_{\mathcal{L}}(\texttt{A}, \texttt{C})^{(\theta, I)}$*. In contrast to a discrete relational feature, one or both of* A *and* agg *in a numeric relational feature must have a continuous range.*

There are a number of aggregation functions one can define and use, but the standard ones used in SRL and in this dissertation are: maximum, minimum, average, and proportion.

Example 26. Consider the following numeric relational feature:

$$average_{\{S\}}(numHours(C), takes(S, C) = true)$$

This feature computes the average number of hours a student spends studying for all taken classes.

Third, the definition of a dependency statement needs to be extended in order to incorporate numeric relational features.

Definition 10. (Hybrid Dependency Statement) *A* hybrid dependency statement *is of the form* G | *Parents*(G) *where* G*'s range may be discrete or continuous and Parents(*G*) is a set of discrete and/or numeric relational features. Each hybrid dependency statement has an associated CPD.*

Note that the type of a CPD for each hybrid dependency is determined according to G's range: for a discrete range it is a probability mass function, and for a continuous range it is a density function.

Now we are ready to formally define an HRDN:

Definition 11. (HRDN) *An HRDN is a tuple* $(\mathcal{P}, RVD, dep)$*, where* $\mathcal{P}$ *is a set of predicates, whose ranges may be discrete or continuous, RVD is a set of randvar declarations and dep is a function mapping each* $\texttt{P} \in \mathcal{P}$ *to a hybrid dependency statement.*

4.2.3 Semantics of HRDNs

The semantics of the random variable declarations and dependency statements in an HRDN is equivalent to that of RDNs. Given a domain, the random variable declarations determine the set of random variables that satisfy the relevancy conditions. The dependency statements define for each random variable which other random variables it depends on.

Analogous to an RDN, an HRDN can be viewed as a template for constructing a hybrid dependency network in the following way. The set of predicates $\mathcal{P}$ in an HRDN is split into the set of predicates with discrete range $\mathcal{P}_D$ and the set of predicates with continuous range $\mathcal{P}_C$. Given a set of random variable declarations *RVD* for all predicates in $\mathcal{P}$ and a set of constants, the set of randvars is $\Phi = \Phi_D \bigcup \Phi_C$ where Φ_D denotes all randvars with discrete ranges and Φ_C denotes all randvars with continuous ranges. The induced hybrid DN will have a node for each randvar in Φ and the parent set of a node is determined in the same manner as described in Section 2.4.2 for discrete DNs. Each discrete randvar of a predicate $\mathsf{P}_d \in \mathcal{P}_\mathcal{D}$ will obtain its own copy of the discrete CPD associated with P_d and each continuous randvar of a predicate $\mathsf{P}_c \in \mathcal{P}_\mathcal{C}$ will obtain its own copy of the continuous CPD associated with P_c.

A consistent HRDN specifies the joint distribution over the randvars in its corresponding hybrid dependency network. In parallel with the claims of Neville and D. Jensen 2007, there is a direct correspondence between consistent HRDNs and hybrid Markov logic networks (HMLN) in that the set of distributions that can be encoded by a consistent HRDN is equal to the set of positive distributions that can be encoded with an HMLN with the same adjacencies provided they use the same aggregate functions. If an HRDN induces a hybrid DN that does not contain cycles, then its semantics corresponds to those of a hybrid Bayesian network. In this dissertation we primarily consider inconsistent HRDNs. In this case, if there is a stationary distribution of an ordered pseudo-Gibbs sampler applied to an HRDN model, we refer to this distribution as the one represented by the model.

The pseudo-loglikelihood of an HRDN is computed as follows:

$$
\begin{aligned}
PLL(M;I) = \sum_{\mathsf{P}_d \in \mathcal{P}_\mathcal{D}} \sum_{g \in gr(\mathsf{P}_d)^I} \log[p(I(\mathrm{g}) \mid I(Parents(\mathrm{g})))] + \\
\sum_{\mathsf{P}_c \in \mathcal{P}_\mathcal{C}} \sum_{g \in gr(\mathsf{P}_c)^I} \log[p(I(\mathrm{g}) \mid I(Parents(\mathrm{g})))].
\end{aligned}
\tag{4.1}
$$

where the first summation goes over the predicates with a discrete range, and the second goes over the predicates with a continuous range.

Example 27. *To illustrate an HRDN, we could extend Example 16 with the* `numHours/1` *predicate and obtain the following domain:*

$$\mathtt{random(intelligence(S))} \leftarrow \mathtt{student(S)}$$

$$\mathtt{random(takes(S,C))} \leftarrow \mathtt{student(S),course(C)}$$

$$\mathtt{random(grade(S,C))} \leftarrow \mathtt{student(S),course(C)}$$

$$\mathtt{random(difficulty(C))} \leftarrow \mathtt{course(C)}$$

$$\mathtt{random(numHours(C))} \leftarrow \mathtt{course(C)}$$

To the discrete dependency

$$\mathtt{grade(S,C)} \;\left|\; \begin{array}{l} \mathtt{value(intelligence(S),}\emptyset\mathtt{),} \\ \mathtt{value(difficulty(C),}\emptyset\mathtt{)} \end{array}\right.$$

we can add the following hybrid dependency statement:

$$\mathtt{numHours(C)} \mid \mathtt{value(difficulty(C),}\emptyset\mathtt{)}$$

which states that the number of hours spent studying for a class depends on its difficulty. Figure 4.1 shows the ground hybrid DN for this example. Squares denote randvars with a discrete range and ovals denote randvars with a continuous range.

4.2.4 Local Distributions for HRDNs

Each dependency statement `G` | Parents(`G`) has an associated CPD. The type of model used for a CPD depends on both the range of the target atom `G` and

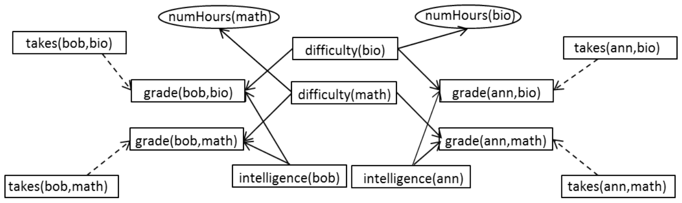

Figure 4.1: The ground HRDN specified in Example 27. Squares represent randvars with a discrete range, and ovals represent randvars with a continuous range. The dashed arrows specify the relevancy condition on `grade/2`.

whether Parents(G) contains discrete or numeric features.

In this work, we use a *parametric* approach to density estimation and focus only on variants of Gaussian distributions to model continuous variables. Specifically, we use the following models:

Multinomial If G has a discrete range and its parent set is empty, the CPD is modeled by a multinomial distribution.

Gaussian If G has a continuous range and its parent set is empty, the CPD is modeled by a Gaussian distribution.

Logistic Regression (LR) This CPD is used when the target atom has a discrete range as it facilitates incorporating both discrete and continuous parents Bishop 2006. Note that this CPD was introduced in Section 2.2.2. However, the difference lies in the fact that the predictors in HRDNs are hybrid relational features. Hence, for the sake of clarity and completeness we give a detailed representation of this CPD.

Given $range(\mathtt{G}) = \{y_1, y_2, ..., y_m\}$, the conditional distribution for the first $(m-1)$ values for a specific grounding $\mathtt{G}\theta$ is:

$$p(\mathtt{G}\theta = y_k \mid Parents(\mathtt{G}\theta)) = \frac{exp\left(w_{k,0} + \sum_{f \in Parents(\mathtt{G})} w_{k,f} \cdot f(\theta)\right)}{1 + \sum_{j=1}^{m-1} exp\left(w_{j,0} + \sum_{f \in Parents(\mathtt{G})} w_{j,f} \cdot f(\theta)\right)}$$

The distribution for the m^{th} value is:

$$p(\mathtt{G}\theta = y_m \mid Parents(\mathtt{G}\theta)) = \frac{1}{1 + \sum_{j=1}^{m-1} exp\left(w_{j,0} + \sum_{f \in Parents(\mathtt{G})} w_{j,f} \cdot f(\theta)\right)} \tag{4.2}$$

In both equations, f is a relational feature, $f(\theta)$ denotes the value of the feature f calculated for the substitution θ, $w_{j,f}$ are the weights associated with f for value y_j, and $w_{j,0}$ is y_j's bias term.

Linear Gaussian (LG) A linear Gaussian CPD is used when $\mathtt{G}$'s range is continuous and all the features in the parent set are numeric (S. L. Lauritzen 1992; Koller, Lerner, et al. 1999). An LG is a Gaussian distribution that models μ as a linear combination of the values of the features in the parent set, but assumes a fixed variance σ^2. The distribution is given as:

$$p(\mathtt{G}\theta \mid Parents(\mathtt{G}\theta)) = N\left(w_0 + \sum_{f \in Parents(\mathtt{G})} w_f \cdot f(\theta), \sigma^2_{\mathtt{G}}\right) \tag{4.3}$$

where f is a numeric feature and w_f is the weight associated with f.

Conditional Linear Gaussian (CLG) A conditional linear Gaussian (CLG) is used if $\mathtt{G}$'s range is continuous and its parent set contains a mix of discrete and numeric features. There is a separate linear Gaussian model for every instantiation of the discrete parents. More formally, consider partitioning the parent set of a predicate into the discrete features, $f_{discrete}$, and the numeric features, $f_{continuous}$ and let $\mathcal{D}$ be the Cartesian product of ranges of all features in $f_{discrete}$. Then, the CPD consists of one LG model for each $d \in \mathcal{D}$:

$$p(\mathtt{G}\theta \mid f_{continuous}, d) = N\left(w_{0_d} + \sum_{F \in f_{continuous}} w_{f_d} \cdot f(\theta), \sigma^2_d\right) \tag{4.4}$$

Note that because there is a separate LG for each d, each one has an associated variance σ^2_d. A *conditional Gaussian* is a special case of a CLG where the parent set only contains discrete features. Here, a separate Gaussian (mean and variance) is learned for each possible configuration of the parents.

As in the discrete case, it is possible that a feature does not have any groundings. If this occurs and the aggregation function of the feature is not defined on the empty set, then we return the value *undefined*.

4.2.5 Probabilistic Inference for HRDNs

Similar to RDNs, inference in HRDNs can be performed by using an ordered pseudo-Gibbs sampler. The difference lies in the fact that HRDNs contain both conditional density functions and probability distributions. Given an HRDN, a set of constants for each type, and possibly a set of relevance conditions, inference is performed as follows.

First, the model is grounded to create the corresponding propositional hybrid dependency network. Second, each randvar gets its own copy of a CPD associated to its predicate. Third, an ordering over the atoms is determined based on the relevance conditions, if specified. This ordering has to ensure that when performing sampling for an atom `H` we first sample the values of the atoms in ξ of the relevance condition `H` $\Leftrightarrow \xi$.

Example 28. *For example, consider the relevance condition:*

$$\texttt{grade(S,C)} = \texttt{not\_relevant} \Leftrightarrow \texttt{takes(S,C)} = \mathit{false} \tag{4.5}$$

In each Gibbs sweep, before we sample values for `grade/2` *we make sure that the values for* `takes/2` *are sampled.*

Finally, in each Gibbs sweep we visit each ground atom in order and resample its value according to its probability distribution or density function. A randvar is assigned a value from its range or obtains the value `not_relevant` if there exists a relevance condition that is satisfied in the sweep. Each sweep results in an interpretation I and a sample corresponds to only the relevant randvars in I.

4.3 Related Work

There are a number of probabilistic formalisms such as hybrid Bayesian networks (Murphy 1998) and hybrid dependency networks (Dobra 2009) that model uncertainty for both continuous and discrete variables but not relations. A commonly used type of hybrid BNs are *conditional linear Gaussian (CLG) networks* (S. Lauritzen and Wermuth 1989), where the conditional distribution of the continuous variables given an assignment to the discrete variables is a multivariate Gaussian. However, CLG networks do not allow for discrete variables to depend on continuous ones. This shortcoming is overcome in *augmented* CLGs (Lerner et al. 2001) that use *softmax* CPDs to represent the dependency of a discrete variable on continuous variables together with numerical integration used within the inference algorithm.

On the one hand, there exist a number of SRL approaches such as logical Bayesian networks (LBNs) (Fierens, Blockeel, Bruynooghe, et al. 2005), probabilistic relational models (Getoor, Friedman, et al. 2001), and relational dependency networks (Neville and D. Jensen 2007) that capture both structure and uncertainty in problems but are generally restricted to discrete data. On the other hand, there are several SRL formalisms that can represent hybrid relational domains including hybrid Markov logic networks (HMLNs) (J. Wang and Domingos 2008), hybrid ProbLog (HProbLog) (Gutmann, Jaeger, et al. 2011), continuous Bayesian logic programs (CBLPs) (Kersting and De Raedt 2001), learning modulo theories (LMT) (Teso et al. 2013) and hybrid probabilistic relational models (HPRMs) (Narman et al. 2010). Additionally, formalisms such as relational continuous models (RCMs) (Choi et al. 2010), Gaussian logic (Kuželka et al. 2011), Poisson dependency networks (Hadiji et al. 2015) can model domains that exclusively contain continuous variables. Next, in terms of representation, we provide a more detailed comparison between our approach and HMLNs, HProblog and CBLPs.

HMLNs, CBLPs and HRDNs all serve as template languages for constructing a different type of propositional graphical model: hybrid Markov networks by HLMNs, hybrid Bayesian networks by CBLPs, and hybrid dependency networks by HRDNs. Hence, each formalism inherits the strengths and weaknesses of the underlying formalism. In contrast, HProblog is a probabilistic extension of Prolog containing also continuous probabilistic facts. There are differences in how each formalism models continuous variables. HRDNs, HProblog and CBLPs explicitly state the form of the distribution (e.g., a Gaussian) and its parameters (e.g., the mean and variance). For example, in HProbLog the fact $(\texttt{X}, gaussian(2,8))$ :: `temp(D, X)` declares the temperature for day `D` to be Gaussian distributed with mean 2 and standard deviation 8. A similar representation of continuous variables is done in CBLPs. In contrast, HMLNs express numeric variables through a set of soft constraints with a Gaussian penalty for diverging values. For example, numeric terms $\texttt{SegType(s, Door)} \cdot (\texttt{Length(s)} = DoorLength)$ have value 0 if the segment is not a door and $-(\texttt{Length(s)} - DoorLength)^2$ otherwise. One notable difference between HRDNs and CBLPs is that CBLPs do not permit a discrete variable to have a continuous parent, whereas this is possible in HRDNs.

None of the mentioned approaches above support structure learning. In Chapter 5 we propose an algorithm for learning the structure of HRDNs. In the following section we briefly provide a scheme for learning HRDNs.

4.4 Towards Learning Hybrid Relational Dependency Networks

We will now provide an overview on learning the structure of HRDNs from data. We will start by introducing the learning task and the format of the data we learn from, and then we give a short overview of our learning strategy.

When performing the **learning task** we assume that the random variable declarations are given. The goal is to learn the hybrid dependency statements and their associated CPDs that maximize a specific scoring criterion. We optimize pseudo-loglikelihood in Equation 4.1 when learning HRDNs. This score has already been used for learning other SRL approaches such as MLNs (Richardson and Domingos 2006). This score is *decomposable* meaning that the contribution for each variable is conditioned on all other attribute values in the data, which allows maximizing the pseudo-loglikelihood for each variable independently. To avoid *overfitting* each CPD can be penalized with a factor that takes into account that the number of parameters needed to encode the CPD is not large.

The **data** we use for learning is in the format of *mega examples* (De Raedt and Kersting 2008; Fierens, Blockeel, Bruynooghe, et al. 2005). We refer to these mega examples as *interpretations* or the sets of true ground facts that describe possible states of the world. For example, in the university domain each interpretation would be one particular collection of students, professors, and courses together with their properties and relations. The facts of probabilistic predicates with Boolean range are specified with `predicate_name(`t_1`,`t_2`)`, where t_1 and t_2 are constants. We make a closed-world assumption (Genesereth and Nilsson 1987), that is, the groundings of `predicate_name` not specified explicitly in the interpretation are considered *false*. The facts of probabilistic predicates with a non-Boolean range are specified with `predicate_name(`t_1`,`t_2`,`Val`)`, where t_1 and t_2 are constants and *Val* is a value assigned to the fact. We consider each of the interpretations to be mutually independent. We also assume *complete data*, which means that we observe a value for each random variable. We leave learning HRDNs from incomplete data as the future work and we discuss it in more detail in Section 7.2.

Example 29. *Consider the university domain consisting of students, courses, professors and random variable declarations introduced in Example 27. For the following domain*

```
student(bob).   course(bio).
student(ann).   course(math).
```

one possible interpretation for the random variables could be:

```
intelligence(bob,105.0).   takes(bob,bio).
intelligence(ann,120.0).   takes(bob,math).


grade(bob,bio,low).     takes(ann,math).
grade(bob,math,high).   takes(ann,bio).


grade(ann,bio,med).     difficulty(bio,med).
grade(ann,math,high).   difficulty(math,high).
numHours(math,100.0).   numHours(bio,80.0).
```

The **learning strategy** is quite straightforward: for each predicate learn a CPD that maximizes the pseudo-loglikelihood. The CPDs represent a set of regression of classification models depending on whether we learn a CPD for a predicate with a discrete or a numeric range. Each hybrid dependency is learned by performing a heuristic search and each iteration of the search involves learning and scoring its associated CPD. When the score does not improve, the search stops. We give a more detailed structure learning procedure in the next chapter. A CPD can be represented in different forms, and two most common forms in SRL are CPTs and relational probability trees. In Chapter 6 we give a more detailed description on how to learn relational probability trees for learning the structure of dynamic HRDNs.

4.5 Conclusions

In this chapter we introduced the formalism of hybrid dependency networks (HRDNs), an upgrade of relational dependency networks to hybrid domains.

We demonstrated how to extend the syntax and semantics of RDNs to hybrid domains by proposing a number of conditional probability models applicable to hybrid relational domains.

5

Structure Learning of Hybrid Relational Dependency Networks

"This time next year we'll be millionaires"
Del Boy, Only Fools and Horses

5.1 Introduction

In this chapter we present our algorithm for learning the structure of hybrid relational dependency networks (HRDNs), which were introduced in Chapter 4. Recall that relational dependency networks enable a structure learning approach that is efficient and easy to parallelize. Because RDNs represent an approximate model this means that the CPDs need not to factor the joint probability distribution. This property permits using a decomposable score function, such as *pseudo-loglikelihood* to evaluate candidate structures. Thus the problem can be tackled by independently learning a locally optimal CPD for each predicate. Therefore, we refer to our approach as the *learner of local models (LLM)*.

All the components needed for accomplishing structure learning in hybrid SRL are included in this chapter. We discuss how we create the *candidate feature space* consisting of *hybrid relational features*. This space is traversed by performing a *search* strategy through the space of candidate HRDNs. Each proposed structure needs to be quantified by means of *parameter estimation* and then *scored* to estimate how well it fits the data. We evaluate our approach on one synthetic and one real-world dataset to evaluate if it is more beneficial to learn in hybrid domains directly instead of discretizing them prior to learning.

5.1.1 Contributions and Bibliographical Note

The first contribution of this chapter is the learner of local models (LLM) algorithm for structure learning of HRDNs. To the best of our knowledge, this is the first attempt to perform structure learning in relational hybrid domains. The second contributions is an experimental evaluation of the proposed approach on two hybrid relational domains. We compare LLM to a learner of Markov logic networks, a version of LLM applied to discretized domains, and to two propositional learners.

This chapter is follow up on the previous chapter and is also based on the following published work:

> *Irma Ravkic, Jan Ramon, Jesse Davis (2015) "Learning relational dependency networks in hybrid domains". In: Machine Learning Journal 2015, volume 100, pp. 217 - 254*

This work was also submitted as a student poster to Intelligent Data Analysis (IDA) 2014, which won the *best video prize*.

The author contributed with the algorithm for learning the structure of RDNs in hybrid domains and an experimental evaluation that shows the benefit of learning in hybrid domains compared to learning in discretized domains.

5.1.2 Chapter Structure

This chapter is structured in the following way. In Section 5.2 we give a high-level control structure for the proposed LLM algorithm. We then demonstrate the experimental methodology and results in Section 5.3.

5.2 The Learner of Local Models (LLM)

Algorithm 7 outlines LLM and it receives as input a set of predicates $\mathcal{P}$, a set of training interpretations D, and a set of validation interpretations V. LLM assumes fully-observed data. At a high level, the algorithm is quite simple. For each predicate $\mathsf{P} \in \mathcal{P}$, it invokes the *LearnOneModel* function to learn a local distribution that models P using $\mathcal{P}$. By using a decomposable score function, such as pseudo-loglikelihood, the global score can be optimized by independently finding the best local distribution for each predicate.[1] The final model M is obtained by conjoining all learned local distributions.

Note that this algorithm has the same high-level control structure as existing approaches for learning RDNs. There are two important differences with existing approaches. The first is that the data may contain continuous variables. The second is that, in order to accommodate dependencies on continuous variables, the local distributions are represented via a logistic regression or a (conditional) linear Gaussian as opposed to a relational probability tree.

Next, we describe in detail how to learn and evaluate local distributions.

Algorithm 7 LLM(Predicates $\mathcal{P}$, Training data D, Validation data V)

```
$M = \{\}$
**for all** $\mathsf{P} \in \mathcal{P}$ **do**
    $CPD_P =$ LearnOneModel($\mathsf{P}, \mathcal{P}, D, V$)
    $M = M \cup \{(P, CPD_{\mathsf{P}})\}$
**return:** $(\mathcal{P}, M)$
```

5.2.1 Learning Local Distributions

Each learned CPD, regardless of its form, in an HRDN is parameterized by a set of features. Learning the structure of the CPD requires determining which features should appear in the parent set. This can be posed as the problem of searching through the space of candidate features. We adopt a *greedy* approach that selects one feature at a time to add to the parent set until no inclusion

[1] Note that because we use greedy search the learned structure is a local and not a global maximum.

improves the score. Thus, in each iteration, the central procedure is finding the single best feature and adding it to the parent set.

We construct candidate features in the following way. First, let $\mathtt{H} = \mathtt{P}(\mathtt{V}_1, \ldots, \mathtt{V}_n)$, where each $\mathtt{V}_i$ is a unique logvar, and let $\mathcal{L} = \{\mathtt{V}_1, \ldots, \mathtt{V}_n\}$. Next, we construct all atoms A such that A is different from H. Then, given a user-defined parameter N, for each A all conjunctions of $k \leq N$ randvar-value tests $\mathtt{C} = \{(\mathtt{G}_1 = \mathtt{v}_1), \ldots, (\mathtt{G}_k = \mathtt{v}_k)\}$ are exhaustively enumerated such that (i) all atoms $\mathtt{G}_i$ have a discrete range, (ii) no atom $\mathtt{G}_i$ is identical to H or A, (iii) the set $Q = \{\mathtt{H}, \mathtt{G}_1, \ldots, \mathtt{G}_k, \mathtt{A}\}$ is connected.[2] These restrictions ensure that the set of candidate features is finite. For each constructed C and A one candidate feature $\mathtt{agg}_{\mathcal{L}}(\mathtt{A}, \mathtt{C})$ for each aggregation function agg applicable to $range(\mathtt{A})$ is generated. We consider the following aggregation functions:

- If no aggregation is needed, we use *value*,
- If $range(\mathtt{A})$ is discrete and not $\{true, false\}$, we use *mode*,
- If $range(\mathtt{A})$ is discrete and $\{true, false\}$, we use *proportion* and *exist*,
- If $range(\mathtt{A})$ is continuous, we use *average*, *maximum*, and *minimum*.

The aggregation function *proportion* computes the proportion of a feature's possible groundings that are true. The other functions take on their traditional meanings.

Algorithm 8 outlines our procedure for learning the dependency for a predicate P. As input, it receives the target predicate P, the full set of predicates $\mathcal{P}$ for the domain, a training set D, and a validation set V. First, the algorithm starts by constructing the set of candidate features for P. Second, it repeatedly iterates through the set of candidate features and evaluates the utility of adding each feature to the parent set. Each feature addition is followed by learning the CPD on the training data D and then scoring it on the validation data V. In each iteration, the single best feature is added to the parent set. If no feature improves the score, the procedure terminates. Note that the form of the CPD depends on both P and the features in the parent set. If P's range is discrete, then the CPD is represented via logistic regression. If P's range is continuous, we use linear Gaussians if the parents only contain numeric features and conditional linear Gaussians when the parent set contains both numeric and discrete features.

[2]Here, we mean connected in the sense that the graph (Q, E) is connected with $E = \{\{u, v\} \mid u, v \in Q \wedge u \text{ and } v \text{ share variables}\}$.

The two following subsections explain how we estimate the parameters of the CPDs using the training data and how we evaluate the local models.

Algorithm 8 LearnOneModel(Target Predicate P, All Predicates $\mathcal{P}$, Training Data D, Validation Data V)

```
$Parents(\mathsf{P}) = \emptyset$
$CPD_{\mathsf{P}} =$ learnCPD($Parents(\mathsf{P}), D$)
$FS =$ GenerateCandidateFeatures($\mathsf{P}, \mathcal{P}$)
**repeat**
    $F_{best} = null$
    $CPD_{best} = CPD_{\mathsf{P}}$

    **for** $F$ in $FS$ **do**
        $CPD_{temp}$ =learnCPD($Parents(\mathsf{P}) \cup \{F\}, D$)
        **if** $score(CPD_{temp}, V) > score(CPD_{\mathsf{best}}, V)$ **then**
            $CPD_{best} = CPD_{temp}$
            $F_{best} = F$

    **if** $F_{best} \neq null$ **then**
        $Parents(\mathsf{P}) = Parents(\mathsf{P}) \cup \{F_{best}\}$
        $CPD_{\mathsf{P}} = CPD_{best}$
        $FS = FS \setminus \{F_{best}\}$
**until** $F_{best} = null$
**return:** $CPD_{\mathsf{P}}$
```

5.2.2 Parameter Estimation for Candidate CPDs

Next, we briefly describe how to estimate the parameters for the CPDs for the different types of dependency statements that may appear in a learned HRDN.

Multinomial The maximum likelihood parameters of the multinomial are learned from the data. For a more detailed description refer to dependency networks in Section 2.2.2.

Gaussian The maximum likelihood estimates $\hat{\theta} = (\hat{\mu}, \hat{\sigma})$ of the Gaussian's parameters $\theta = (\mu, \sigma)$ are obtained from the data in the following way:

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \hat{\sigma} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \hat{\mu})^2 \tag{5.1}$$

Logistic regression Parameter estimation requires learning the weight vectors for the logistic regression model. We follow the standard approach and take the (partial) derivative of the conditional loglikelihood of the data and perform gradient ascent to estimate the weights (Mitchell 1997). For a more detailed description see Section 2.2.2.

Linear Gaussian Parameter learning requires estimating the weight vector for the linear regression model. This can be done via standard techniques for training a linear regressor. We use ridge regression (Bishop 2006). We estimate the variance by computing the expected value of the squared difference between the actual value and the model's predicted value.

Conditional linear Gaussian In CLGs, each configuration of the discrete parents has an associated LG model. The parameters for each LG model are learned as described above.

5.2.3 Evaluating Candidate Models

Traditionally, a candidate model is evaluated using a score function that trades off the model's fit to the data versus some penalty term based on the model's complexity to avoid overfitting. Recall that when learning HRDNs the score of each local structure contributes independently to the score of the global structure. For a candidate global model M and validation data V, we use the following score function, which is based on the *minimum description length (MDL)* (Schwarz 1978):

$$MDL(M,V) = PLL(M,V) - Penalty(M,V) = \tag{5.2}$$

$$= \sum_{\mathsf{P} \in M} LL(m_p, V) - Penalty(m_p, V) \tag{5.3}$$

$$\tag{5.4}$$

where $LL(m_p, V)$ is the score that indicates how well the proposed local structure for predicate P explains the data and $Penalty(m_p, V)$ is the term for penalizing complex structures. Each $LL(m_p, V)$ is optimized independently.

Score

The $LL(m_p, V)$ represents the score for a local structure learned for a specific predicate. More precisely, given a proposed CPD $m = p(\mathsf{P}|Parent(\mathsf{P}))$ and the data V, the score is:

$$LL(m_p, V) = \sum_{I \in V} \sum_{g \in gr(\mathsf{P})^I} \log[p(I(\mathrm{g}) \mid I(Parents(\mathrm{g})))] \tag{5.5}$$

where I is an interpretation assigning values to the random variables, $gr(\mathsf{P})^I$ represents all the random variables in interpretation I, and $I(\mathrm{g})$ represents the value of the random variable g in interpretation I. The probability $p(I(\mathrm{g}) \mid I(Parents(\mathrm{g})))]$ is simply read from the CPD estimated from the training data D. For discrete distributions this is done by reading the probability of $I(\mathrm{g})$ for a given assignment of the parents $I(Parents(\mathrm{g}))$. For continuous distributions the density for a ground atom g is calculated with:

$$p(\mathrm{g}|\mu, \sigma^2) = \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(I(\mathrm{g})-\mu)^2}{2\sigma^2}} \tag{5.6}$$

where μ and σ^2 parameters are obtained as explained in Section 4.2.4. In short, if $p(I(\mathrm{g}) \mid I(Parents(\mathrm{g})))$ represents the linear Gaussian, we calculate the μ as the linear combination of the parent values $I(Parents(\mathrm{g})))$. If it represents the conditional linear Gaussian, we do the same as for the linear Gaussian, but for a particular assignment of the discrete parents $I(Parents_d) \in I(Parents(\mathrm{g}))$.

Penalty

The penalty term $Penalty(m_p, V)$ for a proposed local structure m_p of predicate P in Equation 5.2 is calculated in the following way:

$$Penalty(m_p, V) = \frac{1}{2} \sum_{I \in D} \sum_{\mathsf{P} \in \mathcal{P}} log_2(|gr(\mathsf{P})|^I) \cdot B_{\mathsf{P}} \cdot K$$

where $|gr(\mathsf{P})|^I$ is the number of relevant randvars of predicate P in interpretation I, B_{P} is the number of free parameters in P's CPD and K is the size of P's CPD.[3] Next, we will explain in more detail how B_{P} and K are calculated.

[3]Because we assume that all variables are observed, we do not need to run Gibbs sampling to compute the PLL.

When the CPD for P is represented by a logistic regression model (see Equation 4.2), the number of free parameters is:

$$B_{\mathsf{P}} = (|range(\mathsf{P})| - 1) \cdot (1 + |Parents(\mathsf{P})|)$$

where $(1 + |Parents(\mathsf{P})|)$ is the number of weights that must be learned to parameterize the model (i.e., one for each feature plus the intercept). For continuous CPDs, this is slightly more involved to compute. For a linear Gaussian, the number of free parameters is:

$$B_{\mathsf{P}} = 1 + (1 + |Parents(\mathsf{P})|)$$

where the first 1 is for the variance σ^2 and $(1 + |Parents(\mathsf{P})|)$ is the number of weights that must be learned to parameterize the model (i.e., one for each feature in the parent set plus the intercept). Recall that in a conditional linear Gaussian, one linear Gaussian model is learned for each possible instantiation of the discrete parents. Thus the number of free parameters for a CLG is:

$$B_{\mathsf{P}} = d \cdot (1 + (1 + |Parents_C(\mathsf{P})|))$$

where d is the number of elements in the Cartesian product of the ranges of the discrete parents, $Parents_C(\mathsf{P})$ denotes only numeric features in the parent set of P and $(1 + (1 + |Parents_C(\mathsf{P})|))$ is the number of parameters needed to model each LG.

The size K of P's CPD is the sum of the feature lengths in the parent set:

$$K = \sum_{f \in Parents(\mathsf{P})} |f| \tag{5.7}$$

where $|f| = |\mathsf{agg}_{\mathcal{L}}(\mathsf{A}, \mathsf{C})| = |\mathsf{C}| + 1$ is the length of a feature.

5.3 Experiments

This section empirically evaluates our HRDN structure learning algorithm LLM. Specifically, we want to answer the following questions:

1. How does varying the amount of training data affect the quality of the learned model and the run time of the learning algorithm?

2. Do we learn more accurate models by learning a hybrid model (i.e., explicitly modeling continuous variables) or by discretizing all continuous variables prior to learning?
3. How does our approach compare to MLN (Richardson and Domingos 2006) structure learning?

All our code, data and models are publicly available.[4] We first describe the data sets we will use and then explain the experimental setup. Finally, we present and discuss the results.

5.3.1 Datasets

We use one synthetic and one real-world data set to answer these questions.

Synthetic University Data

We used a modified version of the well-known university model (Getoor, Friedman, et al. 2001) to generate synthetic data described in Example 7. In comparison to the originally proposed model, we made the following alterations. First, we switched the range of `intelligence/1` from discrete to continuous. Second, we added two predicates with continuous ranges: `numHours/1`, which is the estimated number of hours a student needs to study for a course, and `ability/1`, which is the ability of a professor. Finally, we added a Boolean predicate `friend/2`, which denotes whether two students are friends. Appendix A contains a complete description of the model.

We generate synthetic data in two ways. First, we fix the domain size of each type within an interpretation and vary the number of training interpretations. We learn models by using one, two, four, eight and 16 interpretations. We use one validation and one test interpretation. Second, we fix the number of training and validation interpretations to one and vary the domain size of each object. The learned models in this setup are evaluated on a test interpretation consisting of 800 students, 125 courses and 125 professors. Tables 5.1 and 5.2 show the characteristics of the domains for the first and second synthetic setup, respectively.

[4]http://dtai.cs.kuleuven.be/ml/systems/llm

For each experimental condition, we repeat the following process ten times. We generate the appropriate number of interpretations, where each interpretation is constructed by performing 2000 iterations of the ordered pseudo-Gibbs sampling (see Section 6.3) using the handcrafted model and the specified number of constants.

For each generated data set, we also create a corresponding discretized version by binning each continuous randvar into a number of equal-size intervals. We used 2, 4, 6 and 8 bins.

Real-world PKDD'99 Financial Data Set

Our real-world domain is the financial data set from the PKDD'99 Discovery Challenge (Berka 1999). It consists of services one bank offers its clients such as loans, accounts, and credit cards among others. In the original data, the *transaction* table contains more than one million transactions. Therefore, we introduced several predicates (e.g., average of monthly withdrawals for an account) to summarize the information contained in this table. This results in 16 predicates[5] about four types of objects: *clients*, *accounts*, *loans* and *districts*. Ten predicates have a continuous range and six have a discrete range.

We consider *account* to be the central object type in the PKDD'99 financial data set. The original data set consists of 4500 accounts, but we omit ten accounts that have missing data. We then split the data associated with these accounts into ten folds. To avoid leakage of information, all information about clients, loans and districts related to one account appear in the same fold. We used six folds for training, three folds for validation and one for testing. Table 5.3 reports the characteristics of this data set.

Again, we create a discretized version of the data by binning each continuous randvar into a number of equal-size intervals and used 2, 4, 6 and 8 bins.

5.3.2 Experimental Methodology

We compare the following four learners on all experiments:

[5]Table A.2.1 in Appendix A.2 describes the predicates.

#Interpretations	Average Sum of #*Randvars*
1	19,627
2	39,308
4	79,027
8	157,959
16	315,334

Table 5.1: Data set characteristics for the synthetic data when varying the number of interpretations used for learning. #Interpretations is the number of training interpretations. Average Sum #*Randvars* is the number of randvars summed across all training interpretations averaged over the ten generated data sets. Each interpretation has 100 students, 50 courses and 50 professors objects.

LLM-H This corresponds to learning a model using our LLM algorithm on the data containing both continuous and discrete variables.

LLM-D This corresponds to learning a model using our LLM algorithm on the discretized data. Thus each learned local distribution is modeled using a logistic regression CPD.

LSM This corresponds to learning a model using the publicly available implementation of LSM (Kok and Domingos 2010) on the discretized data. LSM is the state-of-the-art Markov logic network structure learning algorithm.

Independent This learner constructs a model on the hybrid data such that all randvars are independent. That is, it models the joint distribution as a product of marginal distributions.

On the experiments involving the PKDD'99 financial data set, we include an additional baseline: a handcrafted model. We built a local model to predict each predicate by a set of handcrafted non-relational features. These features are used to predict a property of an object by means of some other properties of that object. The features can be found in Appendix A.4. For predicates with a discrete range, we used logistic regression. For predicates with a continuous range, we used both linear regression and MP5 (a regression tree) as implemented in Weka (Hall et al. 2009).

$\#Students$	$\#Courses$	$\#Professors$	$Average\ \#Randvars$
100	50	50	19,548
200	75	75	66,577
400	100	100	226,679
800	125	125	796,328

Table 5.2: Data set characteristics for the synthetic data when varying the domain size of each object type in the training interpretation. $\#Students$, $\#Courses$, and $\#Professors$ report the number of each type of object. Average $\#Randvars$ is the number of randvars averaged over the ten data sets generated for each domain size.

Experimental Details

LLM is implemented as a combination of Java and Prolog. Java is used for performing the learning and Prolog is used to compute the value of a feature. When generating features, we set the length of the features to be at most $N = 3$. Usually, in relational domains, only a small fraction of the Boolean atoms is true (e.g., the number of people who are friends is quite sparse compared to the number of possible friendships). Therefore, for efficiency reasons, we subsample the false Boolean atoms during learning (Natarajan, Khot, et al. 2012) to achieve a 1:1 ratio of true to false groundings in all experiments.

For LSM, we contacted the authors in order to know what the most important parameters were to tune. Then, we tried several parameter combinations, and used the validation data to select appropriate ones for each data set.

Evaluation Metrics

We evaluate the quality of the learned models using several metrics. First, to measure the quality of the probability estimates, we report the weighted pseudo-loglikelihood (WPLL) (Kok and Domingos 2005). This corresponds to calculating the PLL of an interpretation as the sum of PLLs for each predicate divided by the number of groundings of that predicate in the interpretation.

Second, to measure the predictive performance, we report the area under the ROC curve (AUC-ROC) for discrete predicates and the normalized root-mean-square error (NRMSE) for continuous predicates. Because we have multi-class categorical variables in our domains, we calculate the multi-class AUC-

#Account	*#Loan*	*#Client*	*#District*	*#Randvars*
4,490	680	5,358	77	3,157,657

Table 5.3: Characteristics of the PKDD'99 financial data set. *#Account*, *#Loan*, *#Client*, and *#District* report the number of objects of each type and *#Randvars* is the number of randvars in the data set.

ROC (Provost and Domingos 2000), which we denote as AUC_{total}. The NRMSE for a predicate ranges from zero to one and is calculated by dividing RMSE by the predicate's range.

Additionally, since we know the model structure for the synthetic data, we compare how closely the learned model reflects the handcrafted structure using the following edit distance. For each predicate, we compare the true parent set to the learned parent set. For each feature in the true parent set, we find its closest feature in the learned parent set according to the following distance metric. The distance Δ between two features, $f_1 = \mathsf{agg_1}_{\mathcal{L}_1}(\mathsf{A}_1, \mathsf{C}_1)$ and $f_2 = \mathsf{agg_2}_{\mathcal{L}_2}(\mathsf{A}_2, \mathsf{C}_2)$, is calculated as:

$$\Delta(f_1, f_2) = |\mathsf{C}_1 \backslash \mathsf{C}_2| + |\mathsf{C}_2 \backslash \mathsf{C}_1| + \delta_{\mathsf{A}_1,\mathsf{A}_2} + \delta_{\mathsf{agg}_1,\mathsf{agg}_2}$$

where $\delta_{\mathsf{A}_1,\mathsf{A}_2}$ equals zero if the two atoms A_1 and A_2 originate from the same predicate and their logvars are equivalent, otherwise it equals one. Similarly, $\delta_{\mathsf{agg}_1,\mathsf{agg}_2}$ equals zero if agg_1 and agg_2 represent the same aggregation function, otherwise it equals one. When the best match is found, both the true and the learned feature are excluded from further comparisons, and the edit distance is incremented by the distance between them. Furthermore, the final distance is incremented by the length of each feature that must be added or removed from the learned dependency parent set.

We use a one-tailed paired t-test to assess the significance of the results obtained through ten independent runs for the synthetic experimental setup and ten folds for the real-world data set. The null hypothesis states that there is no difference between two approaches and we reject it when $p<0.01$. For all metrics, we report the metric itself along with its standard deviation.

	Nr. training interpretations				
	1	2	4	8	16
LLM-H	**-18.22 ± 0.5**	**-18.16 ± 0.5**	**-17.89 ± 0.2**	**-17.87 ± 0.2**	**-17.83 ± 0.3**
LLM-D(#bins=2)	$-21.33 \pm 0.3^*$	$-21.10 \pm 0.3^*$	$-21.06 \pm 0.3^*$	$-21.06 \pm 0.3^*$	$-21.05 \pm 0.2^*$
LLM-D(#bins=4)	$-19.53 \pm 0.6^*$	$-19.27 \pm 0.3^*$	$-19.20 \pm 0.3^*$	$-19.12 \pm 0.3^*$	$-19.04 \pm 0.3^*$
LLM-D(#bins=6)	-19.34 ± 0.9	$-18.77 \pm 0.4^*$	$-18.56 \pm 0.3^*$	$-18.55 \pm 0.3^*$	$-18.52 \pm 0.3^*$
LLM-D(#bins=8)	$-20.03 \pm 1.0^*$	$-19.11 \pm 0.8^*$	$-18.64 \pm 0.5^*$	$-18.62 \pm 0.5^*$	$-18.30 \pm 0.3^*$
LSM(#bins=2)	$-23.33 \pm 0.2^*\dagger$	$-23.28 \pm 0.2^*\dagger$	$-22.86 \pm 0.8^*\dagger$	OoM	OoM
LSM(#bins=4)	$-23.00 \pm 0.3^*\dagger$	$-22.86 \pm 0.4^*\dagger$	$-21.65 \pm 1.3^*\dagger$	OoM	OoM
LSM(#bins=6)	$-22.79 \pm 0.4^*\dagger$	$-22.67 \pm 0.4^*\dagger$	$-22.00 \pm 1.3^*\dagger$	OoM	OoM
LSM(#bins=8)	$-22.62 \pm 0.3^*\dagger$	$-22.62 \pm 0.9^*\dagger$	$-21.27 \pm 1.4^*\dagger$	OoM	OoM
Independent	$-23.68 \pm 0.4^*$	$-23.63 \pm 0.4^*$	$-23.53 \pm 0.4^*$	$-23.54 \pm 0.4^*$	$-23.52 \pm 0.4^*$

Table 5.4: The WPLL on the synthetic data as a function of the number of training interpretations. The best WPLLs are in bold, an asterisk (*) denotes significantly worse results for $p<0.01$ compared to LLM-H. A dagger (†) denotes when LSM performs significantly worse than LLM-D for $p<0.01$ on the data discretized with the same number of bins. OoM denotes out of memory.

5.3.3 Results and Discussion

We now present experimental results for the synthetic and real-world data sets.

Results on Synthetic Data

Table 5.4 shows how the WPLL of each approach varies as a function of the number of training interpretations. Learning from the hybrid data results in a significantly more accurate learned model than learning from the discretized data in all cases except for one in which we have one training interpretation and six discretizing bins. When using the same number of bins for discretization, LLM-D learns more accurate models than LSM on all settings. Note that LSM ran out of memory on all runs when training on eight and 16 interpretations. Finally, all learning approaches always outperform the no-learning baseline.

Table 5.5 presents the run times for all algorithms as a function of increasing the number of training interpretations. LSM is the fastest learner, but it produces lower-quality models. For all approaches, the run time scales linearly with the number of interpretations. Learning an HRDN is always faster than learning an RDN. When discretizing the data, the run time is influenced by the number of bins used: the more bins there are, the slower the discrete learner is. This occurs because adding more bins increases the size of the search space.

	Nr. training interpretations				
	1	2	4	8	16
LLM-H	15.58 ± 1.8	18.72 ± 2.3	28.53 ± 2.7	**48.16 ± 3.2**	**76.51 ± 3.51**
LLM-D(#bins=2)	21.82 ± 4.3	30.48 ± 5.5	53.03 ± 13.7	85.74 ± 10.5	140.62 ± 19.2
LLM-D(#bins=4)	28.71 ± 5.7	40.24 ± 7.3	70.54 ± 17.5	109.80 ± 21.0	162.13 ± 41.0
LLM-D(#bins=6)	35.69 ± 7.1	49.84 ± 9.6	83.34 ± 24.0	135.73 ± 13.1	201.40 ± 139.2
LLM-D(#bins=8)	42.70 ± 8.5	57.63 ± 14.4	98.33 ± 29.1	166.11 ± 38.4	255.90 ± 46.7
LSM(#bins=2)	3.73 ± 0.1	6.70 ± 0.0	**7.48** ± 0.1	OoM	OoM
LSM(#bins=4)	3.44 ± 0.1	**6.22** ± 0.1	8.49 ± 0.0	OoM	OoM
LSM(#bins=6)	**3.22 ± 0.0**	6.23 ± 0.0	12.65 ± 0.0	OoM	OoM
LSM(#bins=8)	4.34 ± 0.1	6.27 ± 0.1	13.33 ± 0.1	OoM	OoM

Table 5.5: The run times in minutes on the synthetic data as a function of the number of training interpretations. The best run times are in bold and OoM denotes out of memory.

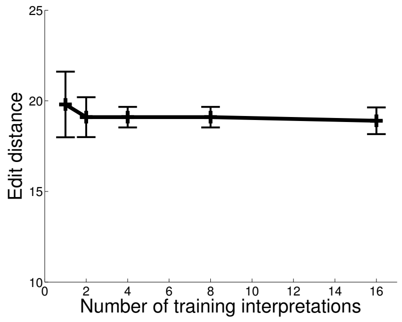

Figure 5.1: The effect of the number of training interpretations on the average edit distance between the handcrafted HRDN model and the hybrid model learned with LLM-H.

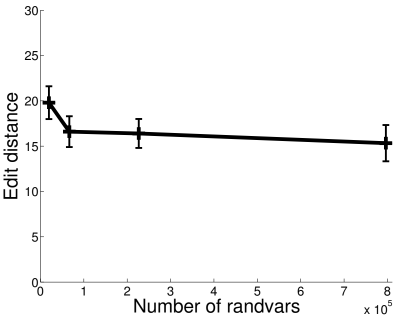

Figure 5.2: The effect of increasing the domain size of each object type on the average edit distance between the handcrafted HRDN model and the hybrid model learned with LLM-H. We summarize the effect of changing the domain sizes by showing the number of randvars in the training interpretation.

Finally, Figure 5.1 shows how the edit distance varies as a function of the number of training interpretations. As expected, the edit distance decreases as more training data are used.

Table 5.6 shows the WPLLs of all learners as a function of increasing the domain size for each object. To encapsulate the effect of domain size changes in a single number, we use the number of randvars in an interpretation. Again, we see that all the learners outperform the independent model. LLM-H always learns significantly more accurate models than LSM. LLM-H learns a significantly more accurate model than LLM-D except when discretizing the data into 6 or 8 bins on the data sets with 200, 400 and 800 students.

Table 5.7 shows the run time of all approaches as a function of increasing domain size. Similar to the previous setup, LSM exhibits better run times than either LLM-H or LLM-D, but it produces lower-quality models. As expected, both LLM-H and LLM-D run time varies quadratically with the increase in domain size. LSM's run time seems to vary linearly, which probably occurs due to its random-walk style search for patterns, which does not necessarily examine all the variables in the training database. When learning (H)RDNs, LLM-H is faster than LLM-D. Again, in general, increasing the number of bins increases the training time.

Figure 5.2 shows that the edit distance between LLM-H's learned model and the handcrafted model decreases as the number of randvars in the training interpretation increases. More (observed) random variables equates to more training data, and, as expected, more data allows us to learn more accurate models.

In both synthetic setups, we noticed that in the learned model `difficulty(C)` depends on `nrhours(C)`. This dependency is not encoded explicitly in the handcrafted model. However, `nrhours(C)` does depend on `difficulty(C)` in the original model. In both cases, this contributes to the edit distance.

More detailed results for both synthetic setups can be found in Appendix A.3.1.

Results on the PKDD'99 Financial Data Set

Figure 5.3 shows the WPLL for all approaches on the PKDD'99 financial data set as a function of the number of bins used for discretization. For the handcrafted models, we denote the combination of logistic regression and linear regression as LR+LinR, and the combination of logistic regression and MP5 regression trees with LR+MP5. In the figure, the lines for LLM-H, LR+LinR, LR+MP5 and the independent model are straight because these approaches operate directly on the hybrid data and hence do not perform discretization. We see a clear ranking between the approaches: LLM-H $>$ LR+LinR $>$ LR+MP5 $>$ LLM-D $>$ LSM $>$ independent.

	Domain size (#students × #courses × #professors)			
	100×50×450	200×75×75	400×100×100	800×125×125
LLM-H	$\mathbf{-18.11 \pm 0.3}$	$\mathbf{-17.84 \pm 0.2}$	$\mathbf{-17.78 \pm 0.2}$	$\mathbf{-17.72 \pm 0.3}$
LLM-D(#bins=2)	$-20.56 \pm 0.5^{*}$	$-20.47 \pm 0.2^{*}$	$-20.42 \pm 0.2^{*}$	$-20.54 \pm 0.2^{*}$
LLM-D(#bins=4)	$-18.95 \pm 0.8^{*}$	$-18.50 \pm 0.2^{*}$	$-18.48 \pm 0.2^{*}$	$-18.60 \pm 0.3^{*}$
LLM-D(#bins=6)	$-18.62 \pm 0.9^{*}$	-18.22 ± 0.6	-17.86 ± 0.2	-17.87 ± 0.2
LLM-D(#bins=8)	$-19.39 \pm 0.8^{*}$	-18.17 ± 0.6	-18.05 ± 0.6	-17.86 ± 0.4
LSM(#bins=2)	$-24.45 \pm 0.2^{*}\dagger$	$-22.58 \pm 0.1^{*}\dagger$	$-22.53 \pm 0.2^{*}\dagger$	$-22.72 \pm 0.2^{*}\dagger$
LSM(#bins=4)	$-23.00 \pm 0.3^{*}\dagger$	$-23.15 \pm 0.5^{*}\dagger$	$-22.20 \pm 0.2^{*}\dagger$	$-21.83 \pm 0.2^{*}\dagger$
LSM(#bins=6)	$-22.79 \pm 0.4^{*}\dagger$	$-25.55 \pm 0.3^{*}\dagger$	$-21.83 \pm 0.2^{*}\dagger$	$-21.92 \pm 0.2^{*}\dagger$
LSM(#bins=8)	$-22.62 \pm 0.3^{*}\dagger$	$-25.64 \pm 0.1^{*}\dagger$	$-21.71 \pm 0.2^{*}\dagger$	$-21.79 \pm 0.2^{*}\dagger$
Independent	$-23.55 \pm 0.1^{*}$	$-23.48 \pm 0.1^{*}$	$-23.46 \pm 0.1^{*}$	$-23.42 \pm 0.1^{*}$

Table 5.6: The WPLL on the synthetic data as a function of the domain size. The best WPLLs are in bold, an asterisk (*) denotes significantly worse results for $p<0.01$ compared to LLM-H. A dagger (†) denotes when LSM performs significantly worse than LLM-D for $p<0.01$ on the data discretized with the same number of bins.

	Domain size (#students × #courses × #professors)			
	100×50×50	200×75×75	400×100×100	800×125×125
LLM-H	15.58 ± 1.8	54.61 ± 3.2	171.82 ± 21.5	2171.05 ± 306.3
LLM-D(#bins=2)	21.82 ± 4.3	98.68 ± 5.0	270.81 ± 38.7	2164.11 ± 154.6
LLM-D(#bins=4)	28.71 ± 5.7	128.51 ± 8.6	341.76 ± 40.7	3026.22 ± 317.4
LLM-D(#bins=6)	35.69 ± 7.1	156.98 ± 8.1	476.16 ± 50.8	2923.45 ± 150.7
LLM-D(#bins=8)	42.71 ± 8.5	182.18 ± 5.8	700.07 ± 80.8	4119.31 ± 387.3
LSM(#bins=2)	3.73 ± 0.1	6.13 ± 0.1	11.85 ± 0.1	**18.34** ± **0.3**
LSM(#bins=4)	3.44 ± 0.1	**5.18** ± **0.1**	**10.32** ± **0.1**	19.72 ± 0.2
LSM(#bins=6)	**3.22** ± **0.0**	5.74 ± 0.1	11.80 ± 0.1	19.68 ± 0.2
LSM(#bins=8)	4.34 ± 0.0	5.71 ± 0.1	11.33 ± 0.1	20.68 ± 0.2

Table 5.7: The run times in minutes on the synthetic data as a function of the domain size for all the learners. The best run times are in bold.

Table 5.8 shows the (multi-class) AUCs and NRMSE for LLM-H and the handcrafted models. All three approaches tend to have similar results on most predicates. Note that the handcrafted features used to propositionalize the data are all features that LLM-H is able to learn automatically.

Table 5.9 reports the AUC_{total} for LLM-H, LLM-D and LSM. Out of the six discrete predicates, LLM-H has a higher AUC_{total} on one predicate, the same on two and worse on three compared to LLM-D. Compared to LSM, it wins on three predicates, loses on two and draws on one.

Figure 5.4 shows the run times for this data set as a function of the number of bins used for discretization. LLM-H exhibits better run times than both LLM-D and LSM. LSM is faster than LLM-D except when discretizing the data into two bins.

When we inspected the models learned on the PKDD'99 financial data set, we found a considerable number of bi-directional dependencies. This means that our algorithm succeeded in learning a model that is mostly structurally consistent. For example, it learned that the monthly payment amount for a loan depends on the loan amount, and vice versa. The same holds for the average salary and the ratio of urban inhabitants in a district, the average amount withdrawn from an account and the average amount credited to an account, the average amount withdrawn from an account and the average number of withdrawals for an account, among others.

Evaluation	Predicate	LR+LinR	LR+MP5	LLM-H
AUC_{total}	`clientDistrict/2`	$0.59 \pm 0.02^*$	$0.59 \pm 0.02^*$	$\mathbf{0.64 \pm 0.02}$
	`gender/1`	$\mathbf{0.50 \pm 0.01}$	$\mathbf{0.50 \pm 0.01}$	$\mathbf{0.50 \pm 0.01}$
	`hasAccount/2`	$0.50 \pm 0.01^*$	$0.50 \pm 0.01^*$	$\mathbf{0.56 \pm 0.01}$
	`freq/1`	$\mathbf{0.86 \pm 0.01}$	$\mathbf{0.86 \pm 0.01}$	$0.82 \pm 0.01^*$
	`hasLoan/2`	$0.76 \pm 0.01^*$	$0.76 \pm 0.01^*$	$\mathbf{1.00 \pm 0.01}$
	`loanStatus/1`	$\mathbf{0.79 \pm 0.03}$	$\mathbf{0.79 \pm 0.03}$	$0.66 \pm 0.04^*$
NRMSE	`clientAge/2`	0.28 ± 0.03	$\mathbf{0.28 \pm 0.01}$	0.28 ± 0.02
	`avgSalary/1`	$0.13 \pm 0.01^*$	$\mathbf{0.11 \pm 0.01}$	$0.13 \pm 0.02^*$
	`ratUrbInhab/1`	$0.20 \pm 0.01^*$	$\mathbf{0.15 \pm 0.00}$	$0.20 \pm 0.00^*$
	`avgSumOfW/1`	$\mathbf{0.02 \pm 0.00}$	$0.03 \pm 0.00^*$	$\mathbf{0.02 \pm 0.00}$
	`avgSumOfCred/1`	0.02 ± 0.01	$0.03 \pm 0.00^*$	$\mathbf{0.02 \pm 0.00}$
	`stdOfW/1`	0.05 ± 0.01	$\mathbf{0.05 \pm 0.00}$	0.05 ± 0.01
	`stdOfCred/1`	0.05 ± 0.01	$\mathbf{0.04 \pm 0.01}$	0.05 ± 0.01
	`avgNrWith/1`	$0.12 \pm 0.02^*$	$\mathbf{0.10 \pm 0.00}$	$0.15 \pm 0.01^*$
	`loanAmount/1`	0.15 ± 0.02	$\mathbf{0.15 \pm 0.01}$	0.16 ± 0.02
	`monthlyPayments/1`	0.17 ± 0.02	$\mathbf{0.17 \pm 0.01}$	0.18 ± 0.02

Table 5.8: The performance of the two variants of the handcrafted models, LR+LinR and LR+MP5, compared to LLM-H on the hybrid data for the PKDD'99 financial data set. LR+LinR uses logistic regression for discrete predicates and linear regression for continuous predicates, and LR+MP5 uses logistic regression for discrete predicates and regression trees for continuous predicates. The best results are in bold and an asterisk (*) denotes the result that is significantly worse ($p<0.01$) than the best result.

More detailed results for the PKDD'99 financial data set can be found in Appendix A.3.2.

Discussion. Now we can revisit and answer the three experimental questions posed at the beginning of this section. To address the first question, we used the synthetic data to explore the scaling behavior of our algorithm. We found that as the amount of training data increases both the accuracy of the learned models and their faithfulness to the ground truth model slightly improve.

The second question revolves around whether it is better to learn from hybrid data or discretized data. On all experiments, we have seen that learning from the hybrid data directly consistently results in significantly more accurate learned models (according to WPLL) than discretizing the data prior to learning. Finally, we wanted to compare our proposed learning algorithm to the state-of-the-art MLN learner. The results show that on both hybrid and discrete data LLM

		Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	LLM-H	LLM-D	LSM	LLM-D	LSM	LLM-D	LSM	LLM-D	LSM
`clientAge/2`	-	0.49 ± 0.04	0.50 ± 0.00	$\mathbf{0.51 \pm 0.04}$	0.50 ± 0.00	0.49 ± 0.03	0.50 ± 0.00	0.50 ± 0.01	0.50 ± 0.00
`clientDistrict/2`	$\mathbf{0.64 \pm 0.02}$	0.56 ± 0.01	0.50 ± 0.00	0.56 ± 0.01	0.50 ± 0.00	0.55 ± 0.01	0.50 ± 0.00	0.56 ± 0.01	0.50 ± 0.00
`gender/1`	$\mathbf{0.50 \pm 0.01}$	$\mathbf{0.50 \pm 0.00}$	$\mathbf{0.50 \pm 0.00}$	$\mathbf{0.50 \pm 0.00}$	$\mathbf{0.50 \pm 0.00}$	$\mathbf{0.50 \pm 0.01}$	$\mathbf{0.50 \pm 0.01}$	$\mathbf{0.50 \pm 0.00}$	$\mathbf{0.50 \pm 0.00}$
`hasAccount/2`	$\mathbf{0.56 \pm 0.01}$	$\mathbf{0.56 \pm 0.01}$	0.50 ± 0.00	$\mathbf{0.56 \pm 0.01}$	0.50 ± 0.00	$\mathbf{0.56 \pm 0.01}$	0.50 ± 0.00	$\mathbf{0.56 \pm 0.01}$	0.50 ± 0.00
`avgSalary/1`	-	0.87 ± 0.00	$\mathbf{1.00 \pm 0.00}$	0.67 ± 0.00	$\mathbf{1.00 \pm 0.01}$	0.59 ± 0.00	$\mathbf{1.00 \pm 0.00}$	0.56 ± 0.00	0.49 ± 0.00
`ratUrbInhab/1`	-	$\mathbf{0.60 \pm 0.00}$	$\mathbf{0.60 \pm 0.00}$	0.52 ± 0.00	0.50 ± 0.00	0.51 ± 0.00	0.50 ± 0.00	0.51 ± 0.00	0.50 ± 0.00
`avgSumOfW/1`	-	$\mathbf{0.99 \pm 0.01}$	$\mathbf{0.99 \pm 0.00}$	0.85 ± 0.02	$\mathbf{0.99 \pm 0.00}$	0.64 ± 0.00	$\mathbf{0.99 \pm 0.00}$	0.63 ± 0.02	$\mathbf{0.99 \pm 0.01}$
`avgSumOfCred/1`	-	$\mathbf{0.99 \pm 0.01}$	$\mathbf{0.99 \pm 0.00}$	0.96 ± 0.01	$\mathbf{0.99 \pm 0.00}$	0.80 ± 0.06	$\mathbf{0.99 \pm 0.00}$	0.66 ± 0.09	$\mathbf{0.99 \pm 0.00}$
`stdOfW/1`	-	0.90 ± 0.02	$\mathbf{0.98 \pm 0.01}$	0.88 ± 0.06	0.97 ± 0.01	0.65 ± 0.05	0.97 ± 0.01	0.60 ± 0.01	0.95 ± 0.01
`stdOfCred/1`	-	0.91 ± 0.04	0.96 ± 0.06	0.81 ± 0.04	0.96 ± 0.03	0.68 ± 0.01	$\mathbf{0.98 \pm 0.01}$	0.64 ± 0.04	0.97 ± 0.01
`freq/1`	0.82 ± 0.01	0.64 ± 0.03	0.86 ± 0.17	0.59 ± 0.05	0.78 ± 0.15	0.57 ± 0.03	$\mathbf{0.88 \pm 0.10}$	0.56 ± 0.03	0.86 ± 0.10
`avgNrWith/1`	-	0.57 ± 0.03	0.58 ± 0.36	0.56 ± 0.03	$\mathbf{0.67 \pm 0.20}$	0.49 ± 0.02	0.59 ± 0.11	0.51 ± 0.01	0.59 ± 0.18
`hasLoan/2`	$\mathbf{1.00 \pm 0.01}$	$\mathbf{1.00 \pm 0.01}$	0.50 ± 0.00	$\mathbf{1.00 \pm 0.01}$	0.50 ± 0.00	$\mathbf{1.00 \pm 0.01}$	0.50 ± 0.00	$\mathbf{1.00 \pm 0.01}$	0.50 ± 0.00
`loanAmount/1`	-	$\mathbf{0.86 \pm 0.05}$	0.78 ± 0.31	0.72 ± 0.02	0.65 ± 0.25	0.53 ± 0.04	0.67 ± 0.19	0.55 ± 0.03	0.50 ± 0.00
`loanStatus/1`	0.66 ± 0.04	0.56 ± 0.05	0.70 ± 0.25	0.59 ± 0.06	0.75 ± 0.33	0.62 ± 0.05	0.72 ± 0.33	0.51 ± 0.07	$\mathbf{0.78 \pm 0.34}$
`monthlyPayments/1`	-	0.67 ± 0.06	$\mathbf{0.77 \pm 0.22}$	0.66 ± 0.03	0.68 ± 0.19	0.66 ± 0.03	0.52 ± 0.06	0.63 ± 0.03	0.50 ± 0.00

Table 5.9: AUC_{total} results for LLM-H, LLM-D and LSM on the six discrete predicates in the PKDD'99 financial data set. The best results are in bold.

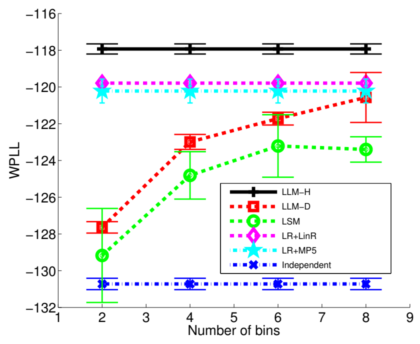

Figure 5.3: The WPLL for each approach on the PKDD'99 financial data set as a function of the number of bins used for discretization. Note that the results for LLM-H, LR+LinR, LR+MP5 and the independent model do not depend on the number of bins used for discretization.

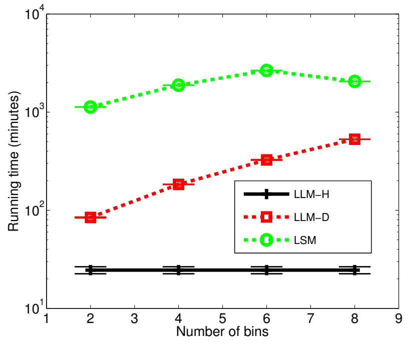

Figure 5.4: The run time of each approach on the PKDD'99 financial data set as a function of the number of bins used for discretization. The y-axis (run time) is on a log scale. Note that LLM-H's results do not depend on the number of bins used for discretization.

learns more accurate models than LSM.

5.4 Related Work

On the propositional level, researchers have considered extending formalisms such as Bayesian networks and dependency networks to model both discrete and continuous distributions. In terms of hybrid Bayesian networks, most of the work has focused on inference (Koller, Lerner, et al. 1999; Yuan and Druzdzel 2007; Murphy 1998; Moral et al. 2001; S. L. Lauritzen and F. Jensen 2001). There have also been some initial attempts for parameter learning (Murphy 1998) and structure learning (Romero et al. 2006). Cobb et al. (2007) provide a more detailed overview of work on hybrid Bayesian networks.

There has been some work on structure learning for hybrid dependency networks. Dobra (2009) has proposed bounded stohastic search for variable selection (structure learning) for sparse genetic dependency networks that contain both discrete and continuous variables. Meinshausen and Bühlmann (2006) use neighbourhood selection with the Lasso for structure learning as a computationally attractive alternative to standard covariance selection methods for multivariate normal distributions. Guo and Gu (2011) use dependency networks for multi-label classification where each CPD represents a probabilistic or non-probabilistic binary classifier that can have both discrete and continuous predictors.

Our work represents a relational approach and builds off of two lines of research: structure learning for RDNs and hybrid relational probabilistic models. There are two existing structure learning approaches for RDNs (Neville and D. Jensen 2007; Natarajan, Khot, et al. 2012). Both approaches perform structure learning by finding the best conditional distribution independently for each predicate. They slightly differ in how they represent the CPDs. Neville and D. Jensen (2007) learn a single relational probability tree (Neville, D. Jensen, et al. 2003) for each predicate. Natarajan, Khot, et al. (2012) represent individual conditional distributions as a weighted sum of relational regression trees (Blockeel and De Raedt 1998), which are learned by a stage-wise optimization procedure. However, these approaches do not explicitly model continuous distributions and instead require them to be discretized. In contrast, our approach is able to directly encode dependencies between discrete and continuous random variables without discretization. Doing so necessitates representing the CPDs with logistic regression or conditional (linear) Gaussian model as opposed to a relational probability tree.

There are several formalisms that focus mostly on representation and reasoning issues in hybrid relational domains including hybrid probabilistic relational models (HPRMs) (Narman et al. 2010) and hybrid problog (HProblog) (Gutmann, Jaeger, et al. 2011). On the other hand, there are several SRL hybrid approach such as hybrid Markov logic networks (HMLNs) (J. Wang and Domingos 2008), continuous Bayesian logic programs (CBLPs) (Kersting and De Raedt 2001) and learning modulo theories (LMT) (Teso et al. 2013) that also support for learning the parameters of a given model from data. Next, we provide a more detailed comparison between our approach and HMLNs, HProblog and CBLPs.

In terms of reasoning, HMLNs and HRDNs use approximate inference. Currently, HProblog only supports an exact inference procedure which involves partitioning the continuous probabilistic facts into admissible intervals. Scaling HProblog to large domains would require the development of a suitable approximate inference algorithm. Inference in CBLPs can be split in two parts: logical inference and probabilistic inference. The former computes the support network for a query (i.e., a Bayesian network containing all relevant variables for the query). The latter applies off-the-shelf Bayesian network inference methods to the resulting support network.

There are significant differences in the level of support for learning in each formalism. Out of the four formalisms, HRDNs are the only one that support structure learning in hybrid domains. Like HRDNs, HMLNs and CBLPs have algorithms for parameter learning. Currently, HProblog does not support parameter learning.

5.5 Conclusions and Future Work

This chapter addressed the problem of learning models from structured, relational data that contain both discrete and continuous variables. To the best of our knowledge, this is the first attempt to perform structure learning in a hybrid SRL setting. We introduced hybrid relational dependency networks (HRDNs), a novel extension of relational dependency networks that accommodate continuous variables and proposed an algorithm that automatically learns the structure of an HRDN from data. Empirically, we evaluated the benefit of incorporating continuous variables in a learned model on one synthetic and one real-world data set by considering two versions of each data set: one that contains both continuous and discrete variables, and one where each continuous variable is discretized prior to learning. We compared our proposed algorithm to two learners that work only on discrete data: a variant of our algorithm and LSM, the state-of-the-art MLN structure learner. We found that learning directly from the hybrid data resulted in more accurate learned models than learning from the discretized data.

One interesting direction for future work is to explore the suitability of modeling other continuous conditional distributions, next to the Gaussians considered in this paper. In principle, other density functions can be used given that we can calculate the value of the function at a point and that we can sample a value for a variable given the assignment to its parents. However, it is unclear how easy this is in practice for complex distributions, and whether issues could arise with sampling inconsistent HRDNs containing relational conditional dependencies. We would also like to extend our learning algorithm such that it could cope with missing data and model latent variables. Additionally, we would like to explore other penalty terms in the objective function such as a L1 penalty that has been used for learning propositional DNs (Dobra 2009; Meinshausen and Bühlmann 2006). Finally, we would like to evaluate our approach on more real-world domains.

6

Learning the Structure of Dynamic Hybrid Relational Models

"These aren't the droids you're looking for."
Obi-Wan, Star Wars

6.1 Introduction

The previous chapter introduced a method for learning the structure of hybrid relational models. However, this approach was designed only for *static relational data* meaning that the training data is a snapshot of the states and relationships between objects at a specific point in time. However, most domains are dynamic and evolve over time. For example, a university is a constantly changing environment, but static models would model only the end of the last semester as the most informative and complete state. On the other hand, *dynamic* models *probabilistically* quantify the regularities that define how the *transition* is made from one state to another. Moreover, it is beneficial for these regularities to capture the *relations* between entities, because objects often interact in dynamic environments. Naturally, planning can be performed when such models are specified: a sequence of actions are chosen based on the dynamic model that will *most probably reinforce* a specific behaviour or task.

Relational Markov decision processes (RMDPs) are a formalism that elegantly combines the principles of SRL with the aspects of probabilistic planning and reinforcement learning in dynamic environments. While RMDPs are popular with planning and learning, their application to domains such as robotics is still severely limited. First, methods for learning the structure of RMDPs are developed on purely discrete domains and *numeric* information is often *discretized* prior to learning. Second, RMDPs are often evaluated on simulated data instead on *real-world robotics scenarios*, which can limit their contribution and applicability.

Thus we adapt the learning approach from the previous chapter to learn a transition model for an RMDP.

6.1.1 Contributions and Bibliographical Note

The contributions of this chapter are the following. First, we propose DDC-TL, a method to learn the structure and the parameters of a hybrid RMDP represented with *dynamic distributional clauses (DDCs)*, which have already been used in a robotics context and for which a planner, called HYPE, exists. We learn DDCs by using LLM, the learner of HRDNs that was introduced in the previous chapter, but upgraded with regression trees for representing the CPDs. Second, we propose using *equational* features defined as mathematical equations applied to the continuous variables in order to model numeric relations between objects. Finally, we demonstrate the utility of our DDC-TL algorithm by applying the learned model to perform planning with HYPE in a simple real-world robotics scenario, and by comparing it to several propositional approaches.

The work in this thesis is based on the following published paper:

> *Davide Nitti, Irma Ravkic, Jesse Davis, Luc de Raedt (2016) "Learning the Structure of Dynamic Hybrid Relational Models" In: 22nd European Conference on Artificial Intelligence (ECAI), (2016), The Hague, the Netherlands*

The contributions of this work are equally shared by the first two authors of the paper mentioned above, in the following way:

- Davide Nitti performed data collection for the robotics scenario with a real robot arm, and did the experiments with the planner and propositional approaches.
- The author of this dissertation mostly focused on the learning part of the contribution by upgrading the LLM to learn with regression trees and equational features, implementing an automated conversion from the learned HRDNs to DDCs, and evaluating the learning performance of DDC-TL.

6.1.2 Chapter Structure

This chapter is structured in the following way. First we introduce our method on how to learn RMDPs in hybrid domains. We then focus on the algorithm for learning relational regression trees with *equational* features, and we introduce continuous probabilistic models used in the leaves. We continue by presenting our experimental setup, results and discussion. We finalize the chapter by providing conclusions.

6.2 State Transition Models in Relational Hybrid Domains

In this dissertation we propose an algorithm (DDC-TL) for learning a state-transition model expressed as a DDC introduced in Section 2.5.2. This model is learned from data described by continuous variables, discrete variables, and relations (e.g., *nextTo*).

Concretely, given a set of discrete-time trajectories $E = (\mathbf{s}_0, a_0, \mathbf{s}_1, a_1, ..., \mathbf{s}_{T-1}, a_{T-1}, \mathbf{s}_T)$, where $\mathbf{s}_t$ is the state (i.e., an interpretation that could describe object properties and relations such as position, orientation, type, and color) and a_t is an action at time t, the goal is to learn DDCs that define a state transition model of the following form:

$$\mathsf{Q}_{t+1} \sim \mathcal{D}(f(\mathbf{s}_t)) \leftarrow a_t, body_t \tag{6.1}$$

Each such clause defines the distribution $\mathcal{D}(f_c(\mathbf{s}_t))$ of a relational random variable $\mathtt{Q}_{t+1} \in \mathbf{s}_{t+1}$ in terms of a set of continuous relational features $f_c(\mathbf{s}_t)$ whenever $\mathtt{body_t}$ holds, where $\mathtt{body_t}$ is a conjunction of literals discrete conditions that refer to the current state $\mathbf{s}_t$ and action a_t. The relational features are essentially the random variables defined in the DDC. Note that DDCs defined in this manner result in a stratification where predicates at time $t+1$ only depend on predicates from the previous time step t. Stratification is required for DDCs to be well-defined (Gutmann, Thon, et al. 2011).

In this dissertation we restrict ourselves to the following choices for specifying the state transition models. The actions for moving objects are *grasping* followed by a vertical or horizontal movement, *pushing* or *tapping*. The features we will be using represent the positions of objects and the derived relations between them. However, unlike related work, concepts like *rightOf*, *closeTo* or *aboveOf* are not manually defined, but are indirectly learned with equational features, which we will describe later. For the actions we assume inverse kinematics and a motion planner available to execute them.

6.2.1 Relational Hybrid Models

The key insight for developing an algorithm for learning DDCs is that we can leverage ideas from the learner of local models (LLM), introduced in Chapter 5, for learning hybrid relational dependency networks (HRDNs) (Ravkic et al. 2015) introduced in Chapter 4.

HRDNs are able to model dependencies in relational domains that have both continuous and discrete variables. Like DDCs, HRDNs use first-order logic as template language for defining conditional probability distributions, and use a set of local distributions to approximate a joint distribution over a set of random variables defined by grounding atoms constructed over a set of predicates $\mathcal{P}$ and constants $\mathcal{C}$. In this paper, we are interested in learning the dependencies that represent Formula (6.1). Hence, $\mathcal{Q}_{t+1}$ denote the set of predicates for which we learn dependencies. We use $\mathcal{P}_t$ to denote the set of predicates describing the previous time step, which are used to construct the features that appear in $\mathit{Parents}(\mathtt{Q_{t+1}})$. Each local distribution is quantified by a dependency $\mathtt{Q_{t+1}} \mid \mathit{Parents}(\mathtt{Q_{t+1}})$, where $\mathtt{Q_{t+1}} \in \mathcal{Q}_{t+1}$ is a predicate and $\mathit{Parents}(\mathtt{Q_{t+1}})$ is a set of relational features that describe how $\mathtt{Q_{t+1}}$ depends on the other predicates in the domain. Such local distributions can be learned using relational regression trees (Blockeel and De Raedt 1998), but unlike standard (relational) regression trees, each leaf does not contain a constant but instead has a linear or logistic regression model. The key observation is that the distribution modeled by a relational regression tree can be represented with DDCs. Therefore, techniques for learning relational regression trees can be adapted for learning (certain classes of) distributional clauses.

Each root-to-leaf path of a relational regression tree can be mapped onto one DDC, where each internal node defines a condition $\mathtt{L}_i$ that appears in the body of the rule or a numerical feature $f \in f_c$, and the leaf contains the probability distribution (density) $\mathcal{D}(f_c)$ that defines the random variable $\mathtt{Q}_{t+1}$ in terms of the features f along the path. We illustrate this in the following example based on a simplified example of a relational regression tree in Figure 6.1, which is learned by DDC-TL.

Example 30. *The example tree in Figure 6.1 corresponds to the following set of DDCs:*

$$
\begin{aligned}
&\mathit{L1} && : \mathtt{pos_x(ID)_{t+1}} \sim \mathtt{gaussian(P + DX, .02)} \leftarrow \\
&\mathit{F1} && : \mathtt{P\ is\ \simeq(pos_x(ID)_t),} \\
&\mathit{F2\text{-}Tap} && : \mathtt{action(ID, tap, DX, DY)_t.}
\end{aligned}
\tag{6.2}
$$

$$
\begin{aligned}
&\mathit{L2} && : \mathtt{pos_x(ID)_{t+1}} \sim \mathtt{gaussian(P, .004)} \leftarrow \\
&\mathit{F1} && : \mathtt{P\ is\ \simeq(pos_x(ID)_t),} \\
&\mathit{F2\text{-}None} && : \mathtt{not(action(ID, Action, DX, DY)_t),} \\
&\mathit{F3\text{-}False} && : \mathtt{not(min\{(\simeq(pos_y(ID)_t) - \simeq(pos_y(ID_2)_t)),} \\
& && \qquad\qquad \mathtt{ID \neq ID_2\} < 0.07).}
\end{aligned}
\tag{6.3}
$$

$$\begin{aligned}
&\mathit{L3} && : \mathtt{pos_x(ID)_{t+1}} \sim \mathtt{gaussian(P + DX, .003)} \leftarrow && (6.4)\\
&\mathit{F1} && : \ \mathtt{P\ is\ } \simeq(\mathtt{pos_x(ID)_t}),\\
&\mathit{F2\text{-}None} && : \ \mathtt{not(action(ID, Action, DX, DY)_t)},\\
&\mathit{F3\text{-}True} && : \ \mathtt{min}\{(\simeq(\mathtt{pos_y(ID)_t}) - \simeq(\mathtt{pos_y(ID_2)_t})),\\
& && \qquad \mathtt{ID} \neq \mathtt{ID_2}\} < 0.07,\\
&\mathit{F4\text{-}True} && : \ \mathtt{min}\{(\simeq(\mathtt{pos_y(ID)_t}) - \simeq(\mathtt{pos_y(ID_2)_t})),\\
& && \qquad \mathtt{ID} \neq \mathtt{ID_2}\} > -0.07,\\
&\mathit{F5\text{-}Tap} && : \ \mathtt{action(ID_2, tap, DX, DY)_t}.
\end{aligned}$$

$$\begin{aligned}
&\mathit{L4} && : \mathtt{pos_x(ID)_{t+1}} \sim \mathtt{gaussian(P, .004)} \leftarrow && (6.5)\\
&\mathit{F1} && : \ \mathtt{P\ is\ } \simeq(\mathtt{pos_x(ID)_t}),\\
&\mathit{F2\text{-}None} && : \ \mathtt{not(action(ID, Action, DX, DY)_t)},\\
&\mathit{F3\text{-}True} && : \ \mathtt{min}\{(\simeq(\mathtt{pos_y(ID)_t}) - \simeq(\mathtt{pos_y(ID_2)_t})),\\
& && \qquad \mathtt{ID} \neq \mathtt{ID_2}\} < 0.07,\\
&\mathit{F4\text{-}False} && : \ \mathtt{not(min}\{(\simeq(\mathtt{pos_y(ID)_t}) - \simeq(\mathtt{pos_y(ID_2)_t})),\\
& && \qquad \mathtt{ID} \neq \mathtt{ID_2}\} > -0.07).
\end{aligned}$$

The aggregation function $\mathtt{min\{U,C\}}$*, where* $\mathtt{U}$ *is a mathematical equation, returns the minimum of the* $\mathtt{U}$*'s values obtained by ranging over possible groundings for the unbound logical variables in* $\mathtt{U}$ *(i.e., logvars not in the target) such that the condition* $\mathtt{C}$ *holds.*

In a nutshell, the next position (axis x) is the current position plus the action displacement in case of a tap action (6.2). If there is no tap action on the object than if its y position is higher (or lower) of 7cm than any other object then the object will *basically not move (clauses* (6.3) *and* (6.4)*). If the object is close to another object that is tapped, than its position will be affected as well (clause* (6.5)*).*

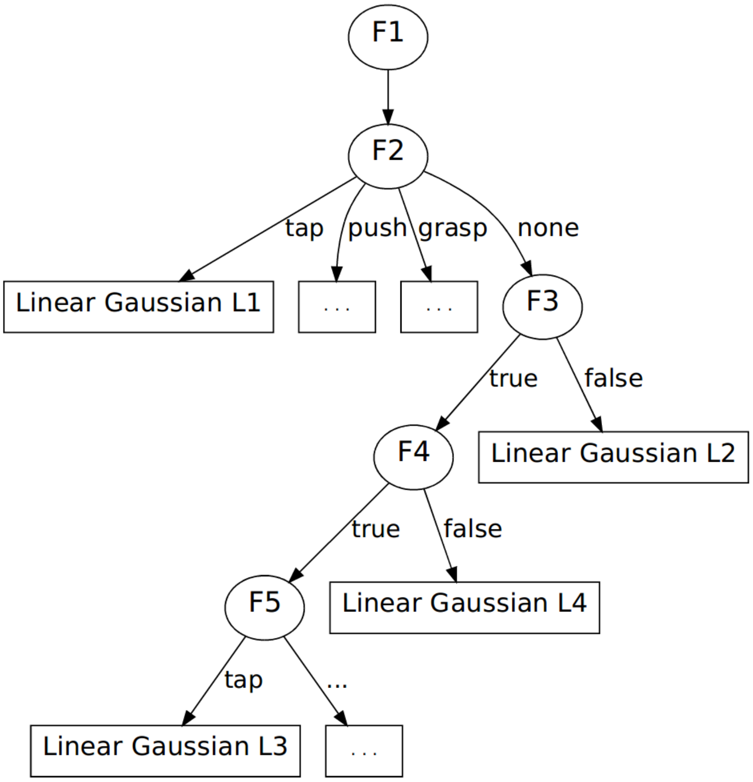

Figure 6.1: A simplified snapshot of relational regression tree for predicting $\texttt{pos(ID)}_{t+1}$. Ellipses represent internal nodes corresponding to learned relational features, and rectangles represent density functions for $\texttt{pos(ID)}_{t+1}$ as presented in Example 30.

One advantage of using relational regression trees to learn DDCs is that they satisfy the mutual exclusiveness property required by DCs, which states that if there are two distributional clauses defining the same continuous random variable, their bodies must be mutually exclusive. This is guaranteed by relational regression trees.

We shall now first introduce the type of features and distributions used by our learning algorithm, and then provide a description of the relational regression tree learning algorithm.

6.2.2 Representing Relational Regression Trees

Each learned DDC will have the form shown in Eq. 6.1 and will correspond to a root-to-leaf path in a relational regression tree. There are two types of internal nodes in such a tree:

- logical conditions which are tests that evaluate to true or false as in standard decision trees. These nodes contribute a condition to $body_t$; and
- continuous features which specify a parameter that appears in the conditional distribution $\mathcal{D}(f_c(\mathbf{s}_t))$ contained in all leaf nodes below this node.

In contrast to standard decision and regression trees, nodes containing a continuous feature do not specify a test. Hence, they do not split the data and only have one child in the tree.

We now define the distributions and features used in our DDC-TL algorithm.

Local Distributions

Leaf nodes, and hence $\mathcal{D}(f_c(\mathbf{s}_t))$ in Eq. 6.1, contain one of the following distributions:

- A **linear Gaussian distribution** defining $\mathcal{D}(f_c(\mathbf{s}_t)) = \mathcal{N}(\mu, \sigma)$ where $\mu = \alpha + \sum_{f \in f_c(\mathbf{s}_t)} f \cdot \beta_f$ if $\mathtt{Q}_{t+1}$'s range is continuous
- A **softmax** or normalized exponential model defining $\mathcal{D}(f_c(\mathbf{s}_t))$ as a $\mathtt{softmax}(\alpha^{(j)} + \sum_{f \in f_c(\mathbf{s}_t)} f \cdot \beta_f^{(j)})$ if $\mathtt{Q}_{t+1}$ ranges over discrete values j, where the $\beta_f^{(j)}$ are the weights for the features $f \in f_c(\mathbf{s}_t)$ and value j in $\mathtt{Q}_{t+1}$'s range.

The set $f_c(\mathbf{s}_t)$ contains the numeric (continuous) features encountered on the root-to-leaf path. Notice that these distributions degenerate into a $\mathcal{N}(\alpha, \sigma)$ and a probability mass function when there are no numeric features, that is, when $f_c(\mathbf{s}_t) = \emptyset$.

Relational Features

Given the (target) random variable $\mathtt{Q_{t+1}}(\mathtt{V_0}, ..., \mathtt{V_n})$ with logical variables $\mathcal{V} = \{\mathtt{V_0}, ..., \mathtt{V_n}\}$, any term $\mathtt{A}$ representing a random variable with logical variables $\forall \mathtt{B_i} \in \mathcal{V}$ can be used as a feature in a distributional clause. For the target random variable $\mathtt{pos_x(ID)_{t+1}}$, any random variable with logical variable $\mathtt{ID}$ is an admissible feature in f_c (if continuous) or in $\mathtt{body_t}$ (if discrete), e.g., $\mathtt{pos_x(ID)_t, pos_y(ID)_t, color(ID)}$. Continuous features can be discretized and added as conditions in $\mathtt{body_t}$, e.g., $\mathtt{pos_x(ID)_t} > 2$.

If the atom $\mathtt{A}$ has some logical variable B_i that does not appear in $\mathcal{V}$, we allow aggregations of the form that was already described in Chapter 4:

$$\mathtt{agg}_{\mathcal{L}}(\mathtt{A}, \mathtt{C})^{(\theta, I)}$$

where agg is an aggregation function, $\mathcal{L}$ is a set of logvars, $\mathtt{A}$ is a term representing a random variable, and $\mathtt{C}$ is a conjunction of atoms that evaluates to true or false (i.e., a condition). The feature computes the specified aggregate agg over values returned as the result of aggregation over all possible groundings of $\mathtt{A}\theta$ that satisfy $\mathtt{C}\theta$ in interpretation I. Sometimes the aggregation might not be defined. For example, if there are no objects satisfying a specific condition $\mathtt{C}$, then aggregation is over an empty set. In case the node represents discretization of an aggregation feature applied to an empty set, the discretized feature is assumed to be *false* as, for example, F3-False in Example 30. If at some node a continuous feature evaluates to $undefined$ during learning, that feature is discarded from the set of candidate features for that node and all its children.

Constructing high-level concepts from low-level numeric sensor data often requires performing mathematical operations (e.g., addition, multiplication, etc.) on the raw data. As our goal is to enable automated discovery of these types of concepts, we introduce *equational* features involving two atoms $\mathtt{A_1}$ and $\mathtt{A_2}$ that both have numeric ranges. These features have the following form:

$$\mathtt{agg}_{\mathcal{L}}\{(\mathtt{A_1}\ \mathtt{op}\ \mathtt{A_2}), \mathtt{C})\}^{(\theta, I)}$$

where op is a mathematical operator $(+, -, *, /)$, and agg, $\mathcal{L}$, and $\mathtt{C}$ are defined as before.

Note that an admissible aggregation over $\mathtt{A}$ needs at least one logical variable $\mathtt{B_i}$ bound with a logical variable in the target $\mathtt{Q_{t+1}}(\mathtt{V_0}, ..., \mathtt{V_n})$, i.e. $\exists i, j : \mathtt{B_i} \in \mathtt{V_j}$.

Example 31. *An example of feature that includes an equation is:*

$$\mathtt{min}\{(\mathtt{pos_y}(\mathtt{ID})_\mathtt{t} - \mathtt{pos_y}(\mathtt{ID_2})_\mathtt{t}), \mathtt{ID} \neq \mathtt{ID_2}\}$$

For a specific grounding of `ID` *bound with the target, this feature calculates the minimum distance in the y-plane between the* `ID` *object and other objects around it. By placing a threshold on this feature, we could distinguish when two objects are close. Thus, these features can represent important concepts that are not explicitly encoded in the raw data.*

The condition `C` *can also be a conjunction, such as in the following feature:*

$$\mathtt{min}\{(\mathtt{pos_y}(\mathtt{ID})_\mathtt{t} - \mathtt{pos_y}(\mathtt{ID_2})_\mathtt{t}), \mathtt{ID} \neq (\mathtt{ID_2}, \mathtt{action}(\mathtt{ID_2}, \mathtt{push}, \mathtt{DX}, \mathtt{DY})_\mathtt{t})\}, \tag{6.6}$$

which calculates the minimum distance between two distinctive objects if the pushing action is performed on the object `ID₂`*. In case this condition is not satisfied the feature value is $false$.*

Logical Conditions

Features can easily be turned into logical conditions. If a feature f is discrete, the condition $f = v$ (with a value v in the range of f) can serve as an atom in the body atoms of a distributional clause. When a discrete feature is included in a node, there will be one subtree for each possible value v of that node in the decision tree, as usual. If f is continuous, we can discretize it by picking some threshold v in the range of f and building a feature such as $f > v$ or $f \leq v$.

6.3 Learning Dynamic Distributional Clauses

In this section we give the algorithm for DDC-TL, our proposed approach for learning state transition models in hybrid relational domains. The algorithm consists of learning a set of relational regression trees for each predicate independently.

Algorithm 9 DDC-TL

```
1: function FeatureSpace($\mathtt{Q}_{t+1}$, $data$, $\mathcal{P}$, $op$, $aggrs$)
2:     $\mathcal{F}_{\mathtt{Q}_{t+1}} \leftarrow$ CONSTRUCT($\mathtt{Q}_{t+1}$, $data$, $\mathcal{P}$, $op$, $aggrs$)
3: function GROWTREE($tree$, $\mathtt{Q}_{t+1}$, $data$, $score$, $\mathcal{F}_{\mathtt{Q}_{t+1}}$)
4:     $f, scoref \leftarrow$ FINDBESTFEATURE($\mathtt{Q}_{t+1}$, $data$, $\mathcal{F}_{\mathtt{Q}_{t+1}}$)
5:     if stopcond($scoref$) then
6:         ADDLEAF($tree$)
7:     if is_continuous($f$) then                                  ▷ Continuous range
8:         $tree \leftarrow$ ADDNODE($f, tree$)
9:         GROWTREE($tree$, $\mathtt{Q}_{t+1}$, $data$, $score$, $\mathcal{F}_{\mathtt{Q}_{t+1}} \setminus f$)
10:    else
11:        $tree \leftarrow$ ADDNODE($f, tree$)                        ▷ Discrete range
12:        for each $a$ in domain($f$) do
13:            $filtered \leftarrow$filter($data, f, a$)             ▷ data such that $f = a$
14:            GROWTREE($tree[a]$, $\mathtt{Q}_{t+1}$, $filtered$, $score$, $\mathcal{F}_{\mathtt{Q}_{t+1}} \setminus f$)
```

Learning Relational Regression Trees

For each predicate $\mathtt{Q}_{t+1}$ occurring in a state, we learn a relational regression tree that defines the distribution according to a set of clauses of the form Eq. 6.1. The algorithm follows a top-down procedure for learning the tree. The outline of the algorithm is provided in Algorithm 9. First, with the function FEATURESPACE it constructs a candidate set of relational features that can appear in the internal nodes when learning the tree for a target predicate Q_{t+1} (how to define legal features will be discussed in detail later). Starting from the empty tree, it adds internal nodes as follows. With FINDBESTFEATURE it iterates through the set of candidate features and tries using each feature as the current internal node. It calculates the difference in score between the new tree and the old tree using five fold internal cross validation on the training data (the score function is discussed in detail later). If no feature improves the score (*stopcond(scoref)* is true), then a leaf node is added. Otherwise, the algorithm greedily selects the highest scoring feature f to include as the internal node. The selected feature is removed from the set of candidate features. If the selected feature has a continuous range and no undefined values in the current branch, the procedure recurses and passes all data to the next node. If the selected feature has a discrete range, one branch is constructed for each value $a \in domain(f)$ of the discrete feature (yielding a logical condition). The data is divided over the branches according to the feature's value, and the procedure recurses along each branch *tree*$[a]$. When no feature improves the score, the recursion stops, a leaf node is added, and the parameters of the leaf's probability distribution or density function are estimated.

Score Function

In Algorithm 9 we use the loglikelihood as the scoring function *scoref*. Basically, the data consists of a set of traces of the form $(\mathbf{s}_0, a_0, \mathbf{s}_1, a_1, ..., \mathbf{s}_{T-1}, a_{T-1}, \mathbf{s}_T)$. Since we are interested in learning the state transition model $p(\mathbf{s}_{t+1}|\mathbf{s}_t, a_t)$ with the Markov assumption, we split the traces into triples of the form $(\mathbf{s}_t, a_t, \mathbf{s}_{t+1})$.

Each triple can be viewed as an interpretation I that assigns values to each of the logical atoms and random variables at time $t+1$ and t appearing in the triplet. The loglikelihood of a triple $I = (\mathbf{s}_t, a_t, \mathbf{s}_{t+1})$ for a set DC of DDC is

$$score(DC, I) = logp_{DC}(\mathbf{s}_{t+1}|\mathbf{s}_t, a_t) =$$

$$\sum_{(\mathsf{H}_{t+1} \sim \mathcal{D} \leftarrow body_t) \in DC} \; \sum_{\theta : body_t\theta \cup \mathsf{H}_{t+1}\theta \subseteq I} logp_{\mathcal{D}}(I(\mathsf{H}_{t+1}\theta)|I(body_t\theta))$$

where $I(\mathtt{q})$ denotes the value assignment of random variable $\mathtt{q}$ in the interpretation I. In this definition we assume that the transition probability factorizes as follows: $p(\mathbf{s}_{t+1}|\mathbf{s}_t, a_t) = \prod_{q_i \in \mathbf{s}_{t+1}} p(q_i|\mathbf{s}_t, a_t)$, i.e. every variable in the next state $q_i \in \mathbf{s}_{t+1}$ depends only on the previous state and action.

Scoring a DDC program first requires estimating the parameters for the CPDs $\mathcal{D}(f_c(\mathbf{s}_t))$. For linear Gaussian distributions, parameter learning requires estimating the weight vector for the linear regression model. This can be done via standard maximum likelihood techniques (e.g., ridge regression (Bishop 2006)). Similarly, softmax parameter estimation requires learning the weight vectors for the logistic regression model. We follow standard gradient ascent approaches to maximize the loglikelihood (Bishop 2006).

Defining Legal Features

We construct candidate features in the similar way as described in Section 5.2, but since we extended the feature space with equational features, we will provide a short description of the full method here.

Feature space in in Algorithm 9 is obtained with the function CONSTRUCT, which receives as input a target predicate $\mathtt{Q_{t+1}}$, a set of aggregation functions (*aggrs*), mathematical operators *op*, and *data* from which the ranges of predicates are extracted. The features of the form $\mathtt{agg}_{\mathcal{L}}(\mathtt{A},\mathtt{C})$ are constructed by exhaustively enumerating all combinations of $\mathtt{agg} \in aggrs$, $\mathtt{A}$, and $\mathtt{C}$ that meet the following constraints. The atom $\mathtt{A}$ does not appear in $\mathtt{C}$, and $\mathtt{C}$ is a conjunction with fewer than $k \leq N$ conjuncts. Each conjunct in $\mathtt{C}$ is either a randvar-value test or an (in)equality constraint between two logvars. Each condition in $\mathtt{C}$ is "linked" to $\mathtt{A}$ via a path of shared logvars, that may pass also through other conditions. As aggregation functions, we consider $\mathtt{min}, \mathtt{max}, \mathtt{avg}$, and $\simeq$. Recall, that $\simeq(\mathtt{A})$ simply returns the value of $\mathtt{A}$ in the data and that it is only applicable if there is exactly one grounding substitution of $\mathtt{A}$ for each grounding substitution θ of $\mathtt{C}$ for which $\mathtt{C}\theta$ is true in the data.

We construct candidate features of the form $\mathtt{agg}\{(\mathtt{U_1}\ \mathtt{op}\ \mathtt{U_2}), \mathtt{C}\}$ in an analogous manner. Both $\mathtt{U_1}$ and $\mathtt{U_2}$ must have continuous ranges. We consider $\mathtt{op} \in \{+, -, *, /\}$ and the aforementioned aggregation functions.

For all features that return a real value, we construct two variants: 1) the feature itself to be used as a parameter of the distribution, and 2) discretized version that can be used as a logical condition in the body of a rule. The second alternative is implemented by discretizing the feature into a number of different bins extracted from the training *data* (in the experiments we used 5 such bins).

Example 32. *To illustrate how to threshold a continuous value to create a discrete feature, consider again the following feature*

$$\mathtt{min}\{(\mathtt{pos_y}(\mathtt{ID})_\mathtt{t} - \mathtt{pos_y}(\mathtt{ID_2})_\mathtt{t}), \mathtt{ID} \neq \mathtt{ID_2}\} \tag{6.7}$$

which calculates the minimum distance in the y-plane between two distinctive objects. In addition to this feature, we also add to the feature space the features with the following template:

$$\mathtt{min}\{(\mathtt{pos_y}(\mathtt{ID})_\mathtt{t} - \mathtt{pos_y}(\mathtt{ID_2})_\mathtt{t}), \mathtt{ID} \neq \mathtt{ID_2}\} \bowtie \mathtt{Thresh} \tag{6.8}$$

where $\bowtie \in \{<, >\}$*, and* $\mathtt{Thresh}$ *is a threshold determined from the training data. If* $\bowtie$ *is* $<$ *and* $\mathtt{Thresh} = 0.07$*, we get Feature F3 from Example 30.*

6.4 Experiments

This section empirically evaluates our approach of learning dynamic hybrid transition model. Specifically we want to answer the following questions:

Q1) How accurate are our learned state transition models for prediction tasks?

Q2) How does DDC-TL compare to the propositional hybrid learners?

Q3) What is the performance of the planner that executes our learned models?

To evaluate these questions, we consider a robotics scenario that consists of a table with cubes and a Kinova MICO robot arm that can manipulate the cubes as shown in Figure 6.2. The learner will only have access to the x, y position of each object. Hence, the learned model must automatically discover important intermediate concepts like `closeTo` by constructing features that build upon the low-level positional data. The learned model is then used for planning.

The learning was performed on Intel(R) Xeon(R) CPU 2.40GHz machines with 128 Gb memory. The planning was performed on a laptop Intel(R) i7 CPU 2.40GHz with 4 Gb memory.

6.4.1 Experimental Setup and Data Generation

To generate data, a sequence of actions is randomly generated and executed on the objects and their effects (positions) are stored, that is, $(\mathbf{s}_t, a_t, \mathbf{s}_{t+1})$ triples. The state is the x, y position of each object. The number of objects used in the experiments are two, three and four. The actions considered are grasping followed by moving horizontally, grasping followed by moving vertically, pushing and tapping. Each action refers to an object and is parameterized by the displacement, a continuous value. The action might fail, e.g., when the object is out of the range of the robot, or the object is too close to other objects. In such cases, the objects will not move or move differently than expected from a successful action execution. The concept of failure is not explicitly encoded and the learner must learn how to distinguish the different effects according to the state and the action.

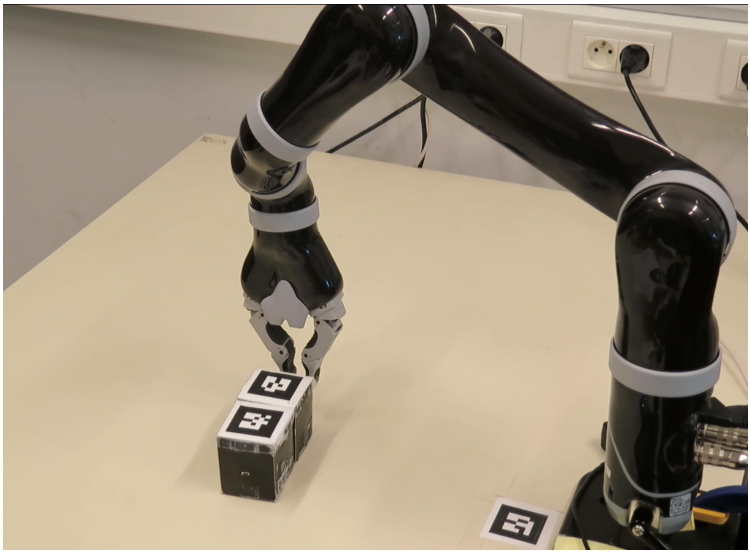

Figure 6.2: MICO arm performing a left tap action on the object on the right. This action moves the other object.

6.4.2 Methodology

We perform the experiments with the following learners:

- **Basic Propositionalization.** We propositionalize the state by constructing one feature for each fact. We denote these with B.
- **Advanced Propositionalization.** We propositionalize the state by using exactly the candidate features considered by DDC-TL. We denote these with A.
- **DDC-TL.** Our approach for learning hybrid dynamic state transition models as introduced in Section 6.2.
- **DDC-TL-50.** A version of DDC-TL using feature selection when learning the tree. This means that when deciding on the split the learner uses only the top 50 selected features.

In Basic Propositionalization, each interpretation with n objects is converted in n training examples, where the target is the next position of an object $\texttt{pos(x)}_{\texttt{t+1}}$, and the features are the positions of all objects at time t. In this case we need to make sure that for each training example with target $\texttt{pos(x)}_{\texttt{t+1}}$, the features corresponding to object x are in the same position.

Example 33. *Given the triplet interpretation* $\{\texttt{pos(1)}_{\texttt{t+1}}, \texttt{pos(2)}_{\texttt{t+1}}, \texttt{pos(3)}_{\texttt{t+1}}, \texttt{pos(1)}_{\texttt{t}}, \texttt{pos(2)}_{\texttt{t}}, \texttt{pos(3)}_{\texttt{t}}\}$ *(the values and the actions are omitted for compactness), the Basic Propositionalization extracts three training examples:*

$$\texttt{pos(1)}_{\texttt{t+1}} \mid \texttt{pos(1)}_{\texttt{t}}, \texttt{pos(2)}_{\texttt{t}}, \texttt{pos(3)}_{\texttt{t}}$$

$$\texttt{pos(2)}_{\texttt{t+1}} \mid \texttt{pos(2)}_{\texttt{t}}, \texttt{pos(1)}_{\texttt{t}}, \texttt{pos(3)}_{\texttt{t}}$$

$$\texttt{pos(3)}_{\texttt{t+1}} \mid \texttt{pos(3)}_{\texttt{t}}, \texttt{pos(2)}_{\texttt{t}}, \texttt{pos(1)}_{\texttt{t}}$$

Note that the feature $\texttt{pos(x)}_{\texttt{t}}$ *of the target object* x *is always the first. The non-target object features do not have a fixed order.*

In DDC-TL we use all the available features for learning. In contrast, DDC-TL-50 uses feature selection in each node to overcome some of the known issues of greedy search and to provide control against overfitting. To select a subset of the features, we use Lasso regression, ARD linear regression, and Random Forests for regression. We pick the top 50 features according to the absolute value of the weights in the linear models, and the feature importance metric of Random Forests.

We use the following propositional learners for our experiments: Lasso, regression trees, and gradient boosted regression trees. We consider three different setups: the data only contains two objects, the data only contains three objects, and the data containing two or three objects. We also evaluate the models learned on two and three objects by applying them to test data that contain four objects. In each case, the goal of the model is to predict the x and y positions of each object in the next time step.

Evaluation

We consider two different evaluation setups. First, we perform 10-fold-cross-validation, and report the average root-mean-squared-error (RMSE) over all held-out folds and learning time. For propositional learners hyper-parameter optimization is performed with internal cross-validated grid search in each of the training folds.

Second, we use the DDC-TL models with the best performance on 10-fold-cross-validation to perform planning using HYPE. We use a reward of 100 if the goal is achieved and -1 otherwise, and consider the following goals:

Region This requires moving any object in a specific region (distant around 20 cm from the closest object).

Swap This entails moving an object to the right (or left) side of another object.

The presence of additional objects can make actions fail. Moreover, if an object is out of reach it can be moved only indirectly by acting first on the other objects that can influence its position. For this reason, it is important that the action effects and the interactions between objects are captured in the learned model. For each goal a set of 15 experiments is performed from different starting positions and the average number of steps is provided.

We assume that every action is applicable (i.e., executable), even when the action does not provide an effect. Obviously, the planner will need to select the actions that are useful to reach the goal. Actions that do not produce movement (i.e., fail) will probably not be selected to reach the goal.

Videos of action executed by the robot are available at `https://dtai.cs.kuleuven.be/ml/systems/DC/`

6.4.3 Results and Discussion

Table 6.1 presents the RMSE of the x and y positions for all approaches, and Table 6.2 summarizes the learning time. Note that the basic propositionalization approach is not applicable when combining the two and three objects data as this approach only works for a fixed number of input variables and hence a fixed number of objects. DDC-TL learns a model that has an error smaller than most of propositional models tested (Q1). In the two objects case, DDC-TL-50 beats all the propositional models (Q2). On the three objects setting, DDC-TL has the best result. On the dataset combining the data of both two and three objects, DDC-TL and Gradient Boosting have the same performance. Note that Gradient Boosting has access to the same features as DDC-TL and is an ensemble approach which provides it with an advantage.

	RMSE			
	Num. of objects			
Learners	2	3	$2+3$	$2+3 \to 4$
DDC-TL	0.029	**0.022**	**0.024**	0.023
DDC-TL-50	**0.027**	0.026	0.026	**0.019**
Lasso-B	0.030	0.026	NA	NA
Regression Tree-B	0.037	0.030	NA	NA
Gradient Boosting-B	0.029	0.025	NA	NA
Lasso-A	0.031	0.026	0.027	0.023
Regression Tree-A	0.037	0.029	0.032	0.030
Gradient Boosting-A	0.029	0.024	**0.024**	0.023

Table 6.1: The RMSEs of predicting the next x and y positions, based on the learned models, averaged over 10 folds.

	Runtime (seconds)		
	Num. of objects		
Learners	2	3	$2+3$
Feature computation	1072.5	2099.5	3186
DDC-TL	3501.5	58322.0	100159.0
DDC-TL-50	1436.0	14846.5	25646.0
Lasso-B	1.8	2.7	NA
Regression Tree-B	0.2	0.7	NA
Gradient Boosting-B	19.9	75.2	NA
Lasso-A	33.7	98.1	160.6
Regression Tree-A	7.2	19.6	26.6
Gradient Boosting-A	636.5	1824.5	2490.5

Table 6.2: The learning time measured in seconds for learning the models for predicting the next x and y positions averaged over 10 folds. Feature computation is the time DDC-TL needs to calculate the values of all the features in the feature space. These calculated feature values are also used for learning propositional models.

Moreover, for the propositional models learned with the advanced feature set and DDC-TL, we used the models learned on data about two and three objects data and evaluated them on test data containing a number of objects (four) that was never seen in the training data. The result for this setting is shown in the last column of Table 6.1. The superior performance in this setting shows an important advantage of our approach, which is that it is able to generalize to new scenarios involving a different number of objects.

Table 6.2 shows the learning time. We report the time needed to calculate the values of all the features in the feature space and the time to actually learn the model. The calculated features are also used when learning propositional approaches. The proposed DDC-TL is more complex, thus generally slower than propositional learning methods. However, our approach has not been optimized and there are number of ways to improve performance (e.g., use feature selection, use a beam search, etc.).

The models have been also qualitatively tested on some actions. For example, Figure 6.3 shows the predicted positions of two objects (based on sampling) after a tap action. The visual inspection confirms that the model captures the interactions between objects. In particular, the model contains relevant features such as the closeness of an object to the object being moved (as shown in Figure 6.1) and the concept 'not reachable', that is, the object is far away from the arm.

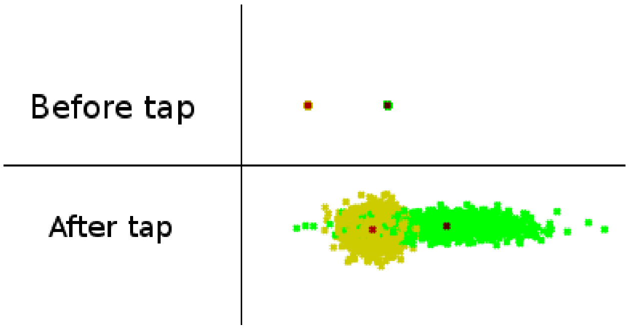

Figure 6.3: Prediction of tap action with two objects. 1000 samples are shown with the mean in dark color.

Table 6.3 presents results about using the learned model with the HYPE planner. For the experiments we use the model learned by DDC-TL with the full feature set. For the **Region** task it has an average success rate of 61% and needs 3.4 steps on average (when it succeeds); whereas for the **Swap** task, it has an

	Avg. # steps (max 5)	Avg. Reward	Success Rate
Region	3.9	56.8	61%
Swap	4.2	57.3	62%

Table 6.3: Planning results using planner HYPE with the model learned by DDC-TL.

average success rate of 62% and needs 3.8 steps on average (Q3). These results confirm that DDC-TL is able to learn a model that is sufficiently accurate for performing simple planning tasks. However, the plans do not always succeed for two reasons. First, there are some scenarios where the objects interactions are not properly modeled. This is expected given that we have a limited amount of training data (196 interpretations for the setup with two objects, and 376 interpretations for the setup with three objects). Second, to keep the planning time reasonable, we limited the number of samples used by the planner to 250. Using more samples could improve the planner's performance.

6.5 Conclusions

To the best of our knowledge, this work represents the first approach that can learn a dynamic statistical relational state transition model in a hybrid domain and then apply an MDP planner to the learned model. The papers central contributions are: adapting HRDN learning for learning dynamic distributional clauses, extending the rich relational feature space to include mathematical equations involving the continuous variables, planning with the learned hybrid models, and identifying a potential robotics application for SRL. Empirically, we demonstrated the merits of our approach using scenario involving a real robotic arm. One of the future directions is to perform learning with partial observability or an unknown number of objects. Another future direction is to explore whether the features that we select in one scenario and that are deemed as interesting can be re-used in future scenarios. This might speed up the learning and increase the expressivity and compression of the models as the number of scenarios grows.

6.6 Related Work

In the literature there are several relational approaches that try to learn a model and use it for planning. However, most approaches only support relational (binary) random variables. For example, Pasula, L. Zettlemoyer, et al. (2007) and Pasula, L. S. Zettlemoyer, et al. (2004) learn noisy indeterministic deictic (NID) rules that define the state in terms of facts (true or false statements). Such representation has been used with the planner PRADA (Lang and Toussaint 2010). Similarly, Moldovan, Moreno, et al. (2012) learn relational affordance models in multi-object manipulation tasks. In such works, the training data is discretized and the model is relational, without the possibility to include continuous variables. This makes the model abstract but useful low level information is lost. Moreover, in robotics applications the discretization is generally handcrafted. In contrast, our approach tries to learn the best features for the prediction task.

Other approaches, based on RL, try to directly learn the Q-function or the policy without learning the state transition model. Among the works with a relational representation there is an approach for performing policy gradient boosting (Kersting and Driessens 2008), and approach for imitation learning (Natarajan, Joshi, et al. 2011). Both methods require a regression tree learner, thus our approach can be easily used in such settings.

Few works consider learning hybrid relational domains. One simplified attempt is the one of Moldovan and De Raedt (2014), that learns the structure of a Bayesian network to model two object effects and convert it into DDC clauses. However, this fixed conversion might not generalize well on more than two objects. In contrast, our approach directly learns a hybrid relational model, this allows to combine data collected with a different number of objects, which provides a better generalization. Moreover, their work considers only simple conditional linear Gaussian models without aggregation or equational features and they evaluate only plans of horizon one.

The work in this chapter is based on the learner of local models (LLM), an approach for learning the structure of HRDNs (Ravkic et al. 2015). Our approach differs from the LLM algorithm used for HRDNs in several crucial ways. Most importantly, we provide support for automatically learning relationships that involve equational features. This is a key capability in terms of being able to model spatial relationships based on low-level continuous data. Additional differences include that we take into account time (and dynamics) and use a tree-based representation for the distributions/densities. Furthermore, the trees are represented using a set of DDCs, enabling their direct use for planning with HYPE.

Beyond incorporating mathematical equations, our tree representation also differs from existing relational regression trees such as TILDE (Blockeel and De Raedt 1998) and its extension towards learning aggregate functions by Van Assche et al. (2006). One difference is that in DDC-TL there is no explicit tying of logvars between different nodes in the tree. We circumvent this by using more complex features that internally support the tying of logvars, e.g., such as Feature 6.6 in Example 31. Another difference is that we use distributions in the leaves. Finally, there may be features on a path that are not used to "split" the data but will be used as features in the distributions.

7 Conclusions

"Live long and prosper"
Spock (Leonard Nimoy)

In this chapter we summarize our main contributions and conclusions. We also provide possible directions for future work.

7.1 Thesis summary

The objective of this thesis was to address several open questions and challenges in statistical relational learning. First, we proposed a *scalable* parameter estimation method that collects the sufficient statistics by approximately counting the number of embeddings of a pattern in a graph that represents relational data. Second, we proposed an *expressive* hybrid SRL formalism for which we also developed a *scalable* structure learning algorithm. And finally, we modelled and learned *expressive dynamic* models in relational hybrid domains, which were then utilized for a real-world robotics planning application.

Contributions

Next, we provide a more detailed description of our contributions:

- Counting the number of occurrences of a graph pattern in a network is in general #P-complete. In this dissertation we described a practical algorithm to **approximately count the number of embeddings of a pattern in graphs** based on the theoretical work of Fürer and Kasiviswanathan (2008). For the theoretical bounds of Fürer and Kasiviswanathan (2008) to hold, the input pattern needs to have an *ordered bipartite decomposition (OBD)*. We also provide two algorithms for obtaining an OBD of a pattern, which was assumed to be given in the theoretical approach. Naturally, we recognized the potential of applying this approach to **parameter estimation in statistical relational models**: counting pattern embeddings is a subroutine for parameter estimation. We performed an extensive experimental evaluation on a wide range of patterns of increasing size and complexity. The experiments showed that even for large patterns the approach always converged and often performed better than the baseline algorithms. Moreover, even for patterns that do not have an OBD the Fürer-Kasiwiswanathan algorithm turned out to perform well.

- Few formalisms in SRL can cope with structured and uncertain data that contain both continuous and discrete variables. For that reason we introduced **hybrid relational dependency networks (HRDNs), which upgrade relational dependency networks to hybrid domains**. Even though they represent an approximation of the joint probability distribution, we advocate using RDNs due to their decomposability property: it is not necessary for the CPDs of an RDN to factor a joint probability distribution, and hence, the CPD for each variable can be learned independently. Hence, each CPD can be optimized independently by means of some regression or classification technique. In this dissertation we proposed to use of several existing CPD models that can accommodate both discrete and continuous variables.

- We proposed **the structure learning algorithm, learner of local models (LLM), for hybrid relational dependency networks (HRDNs)**. We showed the benefit of learning directly in hybrid domains, instead of doing discretization prior to learning. We did our analysis on one synthetic and one real-world data set by considering two versions of each data set: one that contains both continuous and discrete variables, and one where each continuous variable is discretized prior to learning. We compared our proposed algorithm to two learners that work only on discrete data: a variant of our algorithm and LSM, the state-of-the-art MLN structure learner. We found that learning directly from the hybrid data instead of discretizing data prior to learning resulted in a faster structure learning algorithm due to the reduced search space, and more accurate learned models.

- We developed, to the best of our knowledge, the first approach that can **learn a dynamic statistical relational state transition model in a hybrid domain** and then **applied an MDP planner to the learned model**. We accomplished this strategy by adapting HRDN learning to dynamic domains in order to learn dynamic distributional clauses (DDCs) representing relational hybrid MDPs. Moreover we introduced equational features that represent relations between objects defined as arithmetic operators between their low-level attributes. Empirically, we demonstrated the merits of our approach using a scenario involving a real robotic arm. In addition to the novelty of this approach, we showed that the learned models could capture the relationships between objects and the robot arm in such a way that the robot could use the model to perform given tasks by means of planning. We also demonstrated that our approach is able to learn more general models than the propositional approaches we compared to. This means that we were better at predicting the next state of the objects on the data that contains more objects than the data used for learning.

7.2 Future Work

There are still many open questions and challenges in the field of statistical relational learning. In general there is the need for *more expressive* formalims for which structure learning and inference is still efficient and *scalable* when applied to *large scale real-world* domains.

The expressivity can be achieved by choosing richer representation for the models. One approach is performing gradient-based boosting when learning relational dependency networks (Natarajan, Khot, et al. 2012) and Markov logic networks (Khot et al. 2015). The general idea behind this is that each local probabilistic model represents a weighted sum of regression models grown in a stage-wise optimization. The scalability of learning and inference can be achieved with "lifting" (Poole and Zhang 2003) the relational domains: the symmetries present in the domain enable reasoning about groups of objects. There has been some research on *lifted inference and parameter learning* (Ahmadi et al. 2012; Van den Broeck et al. 2013) and *lifted structure learning* (Van Haaren et al. 2015).

Another direction is to do a comparative study of the plethora of formalisms in SRL and to find, combine and exploit their weak and strong points. This would allow one to use efficient methods in one approach and by means of translation obtain another, more adequate or researched formalism. For example, examining the relationships that hold between dependency networks and Markov networks enables the approach of exploiting the efficient but approximate structure learning method of dependency networks and then transforming the learned DNs to Markov networks (Lowd 2012) or Bayesian networks (Hulten et al. 2003). This method should also be upgraded to relational domains.

The general problem in SRL is the lack of the "killer applications" that come in form of large rich real-world relational datasets containing diverse variable types (i.e., discrete and continuous variables) and dynamics. In this dissertation we recognized robotics and its applications as one potential SRL application. However, the trend in performing experiments in robotics, and other similar fields, is usually limited to small-scale simulations or with some simplifications on the expressivity that limit the learning and hence interpretability.

The possible directions for future work we mentioned previously are some general directions in SRL. We next present future work specific to the contributions we presented in this dissertation.

Handling missing data

The algorithms we introduced in this thesis rely on complete data. This means that we do not allow for a random variable to have a missing value. The future work of this dissertation is to investigate how each of the methods presented can be extended to deal with incomplete data.

- **Learning HRDNs with incomplete data** would rely on the approach of *structural expectation-maximization (SEM)* (Friedman 1997) that combines the standard expectation maximization (EM) algorithm, which optimizes parameters, with structure search for model selection. The structural EM has been already applied to some SRL formalisms such as RDNs and MLNs (Khot et al. 2012).

Incorporating continuous representations into structural EM is straightforward, but there are a couple of potential problems with the process. First, there are indications that the EM approach does not behave well with the Gaussian distributions: the likelihood of the data can be arbitrarily large, if the variance of the estimated Gaussians is allowed to go to zero, and extra care needs to be made in this case (Koller, Lerner, et al. 1999). Second, structural EM might not be scalable for HRDNs because completing the data in each iteration of EM requires running a Gibbs sampler. This may be slow and may produce different results depending on the order in which variables are sampled. Alternative approximate approaches such as mean field inference (Lowd and Shamaei 2011) are shown to typically converge much faster than Gibbs sampling and offer similar accuracy.

- **Performing subgraph matching with hidden nodes** is another problem when dealing with missing data. The nodes in a graph can have missing labels, missing values, or both. The missing label problem is more problematic in terms of collecting the expected sufficient statistics than the missing value problem since for the latter we can again use EM for parameter estimation once the embeddings are collected (Luo and Hancock 2001). One possible direction for inferring the expected label while performing matching is to use a so-called *node attendance* (Depiero et al. 1996) by means of relaxation labeling techniques. In this way the label of the data graph is determined based on the consistency of neighboring labels.

- **An unknown number of objects** is a hidden variable that naturally occurs when planning in relational dynamic domains, but is often ignored and a fixed number of objects is assumed. There are a number of challenges when performing learning in this setup. First, if we want to learn the next position of a generic object X given the current state, or more formally $\mathtt{p(pos_{t+1}(X)|s_t, a_t)}$, the next position can depend on the current position $\mathtt{pos_t(X)}$ and other features. However, when the object X is created at time $t+1$ and has not existed in previous time steps, $\mathtt{pos_t(X)}$ and any relation involving X do not exist. For this reason there is no possible binding between logical variables in the body of the rule and `X` in the head. Second, the issue of data association can occur in robotics applications such as object tracking. This entails the uncertainty about the identity of the object when it appears in the scenario: is it the new object *rediscovered* or a *created* object that has never been there before (Vien and Toussaint 2013)? These two options have different representations. Finally, for large robotics domains the learning and planning might not be feasible. One can use the strategy of partitioning the state space into equivalence classes to speed up the learning (Gardiol and Kaelbling 2007).

Re-using and constructing more complex features for hybrid relational MDPs

One interesting direction for learning dynamic hybrid relational models is to extend the learner to compose features out of simpler features that were considered to be important in the past tasks. This resembles predicate or view invention (Davis, Burnside, Page, et al. 2006) in that robots through tasks build a knowledge base with increasing complexity and abstraction levels. A feature can be considered useful if it appears several times as a predictor in the model according to some defined metric. The motivation behind this is to possibly speed up and enrich the learning process as the robot could re-use the knowledge obtained from past scenarios. This aligns in a sense with how humans learn throughout their life and how they acquire and re-use abstractions.

For example, we can learn that the next position of an object depends on the distance between the gripper and the object during pushing actions. Note that this feature is not provided to the learner, but is constructed from primitive spatial features and denotes the aspect we know as the "closeness". From the training data we can learn the following feature for the notion of closeness: $\texttt{close(Obj,Arm)} \leftarrow \texttt{pos}_\texttt{x}\texttt{(Obj)} - \texttt{pos}_\texttt{x}\texttt{(Arm)} < Thr_1$, where $\texttt{pos}_\texttt{x}$ denotes the `x` position of an `object` or the robot `arm`, and Thr_1 denotes some specific threshold value estimated from the data. Once this feature has been added to the background knowledge, it can be used to build more complex features. For example, the object position depends on whether the object has been grasped, i.e. whether `close(Obj,Arm)` and $\texttt{gripper(Arm)} < Thr_2$ hold. This pair of features will probably appear in the learned tree, and the learner can combine them in a new feature. When new features are added, they do not need to be learned again, but can be re-used in similar scenarios.

A Appendix

A.1 Handcrafted and learned HRDNs for the synthetic data

In this “Appendix” we compare the handcrafted and learned hybrid models for the synthetic data set. We present the learned dependencies for both setups: fixed domain size and increasing domain size. For the former we show the learned dependencies when training on 16 interpretations, and for the latter we present the learned dependencies for the largest domain size (800 students, 125 courses and 125 professors).

Predicate declarations

$$range(\texttt{difficulty(C)}) = \{easy, med, hard\}$$

$$range(\texttt{satisfaction(S,C)}) = \{low, med, high\}$$

$$range(\texttt{grade(S,C)}) = \{low, med, high\}$$

$$range(\texttt{takes(S,C)}) = \{true, false\}$$

$$range(\texttt{teaches(P,C)}) = \{true, false\}$$

$$range(\texttt{friend(S,S1)}) = \{true, false\}$$

$$range(\texttt{nrhours(C)}) = [20.0, 180.0]$$

$$range(\texttt{intelligence(S)}) = [50.0, 180.0]$$

$$range(\texttt{ability(P)}) = [20.0, 100.0]$$

Handcrafted model

Below is the model we used to generate the synthetic data.

`difficulty(C)`	
`satisfaction(S,C)`	$\texttt{value}_{\{\texttt{S,C}\}}(\texttt{grade(S,C)},\emptyset)$, $\texttt{value}_{\texttt{C}}(\texttt{ability(P)},\texttt{teaches(P,C)})$
`grade(S,C)`	$\texttt{value}_{\{\texttt{S}\}}(\texttt{intelligence(S)},\emptyset)$, $\texttt{value}_{\{\texttt{C}\}}(\texttt{difficulty(C)},\emptyset)$
`takes(S,C)`	$\texttt{value}_{\{\texttt{S}\}}(\texttt{intelligence(S)},\emptyset)$, $\texttt{value}_{\{\texttt{C}\}}(\texttt{difficulty(C)},\emptyset)$

teaches(P,C)	$\texttt{value}_{\{P\}}(\texttt{ability(P)},\emptyset)$, $\texttt{value}_{\{C\}}(\texttt{difficulty(C)},\emptyset)$
friend(S,S1)	$proportion_{\{S,S_1\}}(\emptyset,\{takes(S,C),takes(S1,C)\})$
nrhours(C)	$\texttt{value}_{\{C\}}(\texttt{difficulty(C)},\emptyset)$
intelligence(S)	$mode_{\{S\}}(\texttt{grade(S,C)},\emptyset)$
ability(P)	

For reasons of reproducibility, we also provide the parameters for the dependencies for our handcrafted model. We will not do this for the learned models as there the parameters are of less interest. The parameters for the dependencies are given in Table A.1.1. The probabilities of multinomial values are to be read in the order given in the predicate declaration. For the logistic regression of a dependency P Parents(P), we use the notation $\texttt{P} = k \rightarrow (w_{k,0}, w_{k,f_1}, ..., w_{k,f_n})$ where $w_{k,0}$ represents the bias term, and w_{k,f_i} represents the ith feature's weight. The order of the feature parameters follows the order of the features in the dependencies. The parameters of conditional Gaussians are of the form $d \rightarrow N(\mu_d, \sigma_d)$, where d represents an instantiation of discrete parents, and $N(\mu_d, \sigma_d)$ gives the Gaussian distribution for that instantiation.

Predicate	Local Distribution	Parameters
`difficulty/1`	Multinomial	(0.2, 0.4, 0.4)
`satisfaction/2`	Logistic Regression	`satisfaction(S,C)`=low →(-1.28,-0.07,0.028) `satisfaction(S,C)`=med→(0.5,-0.4,0.009)
`grade/2`	Logistic Regression	`grade(S,C)`=low →(-1.77,-0.04,1.75) `grade(S,C)`=med →(-2.18,0.003,0.75)
`takes/2`	Logistic Regression	`takes(S,C)`=true →(0.4,0.009,-0.607)
`teaches/2`	Logistic Regression	`teaches(P,C)`=true →(-0.089,-0.012,0.305)
`friend/2`	Logistic Regression	`friend(S,S1)`=true →(-0.08,1.5)
`nrhours/1`	Conditional Gaussian	`difficulty(C)`=easy → N(20,6) `difficulty(C)`=med → N(50,5) `difficulty(C)`=hard → N(80,6)
`intelligence/1`	Conditional Gaussian	`grade(S,C)`=low → N(60,5) `grade(S,C)`=med → N(90,7) `grade(S,C)`=high → N(110,5)
`ability/1`	Gaussian	N(70,10)

Table A.1.1: Local distributions used for the handcrafted model.

Learned model for a fixed domain size (16 training interpretations)

`difficulty(C)`	$\texttt{value}_{\{\texttt{C}\}}(\texttt{nrhours(C)},\emptyset)$
`satisfaction(S,C)`	
`grade(S,C)`	$\texttt{value}_{\{\texttt{S}\}}(\texttt{intelligence(S)},\emptyset)$, $\texttt{value}_{\{\texttt{C}\}}(\texttt{difficulty(C)},\emptyset)$
`takes(S,C)`	$\mathit{proportion}_{\{\texttt{S,C}\}}(\emptyset,\texttt{satisfaction(S,C)=low})$, $\texttt{value}_{\{\texttt{S,C}\}}(\texttt{satisfaction(S,C)},\emptyset)$
`teaches(P,C)`	
`friend(S,S1)`	$\mathit{proportion}_{\{\texttt{S}\}}(\emptyset,\texttt{takes(S,C)})$

nrhours(C)	value$_{\{C\}}$(difficulty(C),$\emptyset$)
intelligence(S)	*mode*$_{\{S\}}$(grade(S,C),$\emptyset$)
ability(P)	

Learned model for a domain size of 800 students, 125 courses and 125 professors

difficulty(C)	*proportion*$_{\{C\}}$($\emptyset$, takes(S,C))
satisfaction(S,C)	value$_{\{S,C\}}$(grade(S,C),$\emptyset$), value$_{\{C\}}$(ability(P), teaches(P,C))
grade(S,C)	value$_{\{S\}}$(intelligence(S),$\emptyset$), value$_{\{C\}}$(difficulty(C),$\emptyset$), value$_{\{S,C\}}$(satisfaction(S,C),$\emptyset$)
takes(S,C)	*proportion*$_{\{S,C\}}$($\emptyset$, {satisfaction(S,C)=mid,friend(S,S1),takes(S1,C)})
teaches(P,C)	value$_{\{P\}}$(ability(P),$\emptyset$)
friend(S,S1)	*proportion*$_{\{S\}}$($\emptyset$, {satisfaction(S,C)=low,grade(S,C)=high, takes(S,C)})
nrhours(C)	value$_{\{C\}}$(difficulty(C),$\emptyset$)
intelligence(S)	*mode*$_{\{S\}}$(grade(S,C),$\emptyset$)

```
ability(P)
```

A.2 PKDD'99 real-world financial data set

Predicate Name	**Description**	**Range**
`clientAge(C,L)`	the age of client `C` at the moment of loan `L` origination	$\mathbb{R}$
`clientDistrict(C,D)`	client `C` lives in district `D`	Boolean
`gender(C)`	the gender of client `C`	{m,f}
`hasAccount(C,A)`	client `C` has an account `A`	Boolean
`avgSalary(D)`	the average salary in district `D`	$\mathbb{R}$
`ratUrbInhab(D)`	the ratio of urban inhabitants in district `D`	$\mathbb{R}$
`avgSumofW(A)`	the average sum of monthly withdrawals for account `A`	$\mathbb{R}$
`avgSumofCred(A)`	the average sum of monthly credits for account `A`	$\mathbb{R}$
`stdOfW(A)`	the standard deviation of monthly withdrawals for account `A`	$\mathbb{R}$
`stdOfCred(A)`	the standard deviation of monthly credits for account `A`	$\mathbb{R}$
`freq(A)`	the frequency of statement issuance for account `A`	{i,d,m}
`avgNrW(A)`	the average number of withdrawals per a month for account `A`	$\mathbb{R}$
`hasLoan(A,L)`	account `A` has loan `L`	Boolean
`loanAmount(L)`	the amount of loan `L`	$\mathbb{R}$
`loanStatus(L)`	the status of loan `L`	{a,b,c,d}
`monthlyPayments(L)`	the monthly payment amount for loan `L`	$\mathbb{R}$

Table A.2.1: Description of the predicates in the PKDD'99 financial data set.

A.3 Detailed results of LLM applied to all domains

In this “Appendix” we present detailed results on per predicate WPLLs for all domains used in our experiments.

A.3.1 Results on synthetic data

Tables [A.3.1-A.3.5] show the test set per randvar WPLLs for each predicate when varying the number of training interpretations. Tables [A.3.6-A.3.9] show the test set per randvar WPLLs for each predicate when varying the domain size of the training interpretations. In both cases, the WPLLs are averaged over all ten runs.

A.3.2 Results on the PKDD’99 financial data set

Tables A.3.10 and A.3.11 contain per randvar WPLLs for all learners applied on the PKDD’99 financial data set. All the WPLLs represent an average value over ten-fold cross-validation.

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
`nrhours/1`	**-4.57**	−5.46	−6.30	−4.90	−6.25	−4.81	−6.11	−5.41	−6.05
`difficulty/1`	**-0.06**	−0.83	−1.59	−0.27	−1.59	−0.28	−1.59	−0.17	−1.59
`ability/1`	**-5.34**	−5.95	−5.95	−5.70	−5.70	−5.71	−5.72	−5.72	−5.67
`intelligence/1`	**-4.89**	−5.71	−7.24	−5.34	−6.09	−5.21	−6.01	−5.39	−5.94
`grade/2`	−1.49	−1.51	−1.53	**-1.48**	−1.53	**-1.48**	−1.53	**1.48**	−1.53
`satisfaction/2`	−1.55	−1.55	**-1.54**	−1.56	**-1.54**	−1.56	**-1.54**	−1.55	**-1.54**
`takes/2`	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**
`friend/2`	**-0.14**	**-0.14**	−0.15	−0.15	−0.15	−0.15	−0.15	−0.15	−0.15
`teaches/2`	**-0.14**	**-0.14**	**-0.14**	**-0.14**	**-0.14**	**-0.14**	**-0.14**	**-0.14**	**-0.14**
Total WPLL	**-18.22**	−21.33	−24.45	−19.53	−23.00	−19.34	−22.79	−20.03	−22.62

Table A.3.1: The per randvar WPLL for each predicate on the synthetic data when training on one interpretation. The best results are in bold.

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
`nrhours/1`	**-4.43**	−5.41	−6.27	−4.82	−6.20	−4.61	−6.09	−5.01	−6.14
`difficulty/1`	−0.18	−0.69	−1.56	−0.28	−1.56	−0.08	−1.56	**-0.10**	−1.44
`ability/1`	**-5.33**	−5.93	−5.96	−5.66	−5.66	−5.67	−5.67	−5.66	−5.72
`intelligence/1`	**-4.87**	−5.71	−7.18	−5.21	−6.08	−5.09	−5.99	−5.01	−5.96
`grade/2`	−1.50	−1.51	−1.53	**-1.46**	−1.53	−1.48	−1.53	−1.47	−1.53
`satisfaction/2`	−1.55	−1.55	**-1.54**	−1.55	**-1.54**	−1.55	**-1.54**	−1.55	**-1.54**
`takes/2`	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**
`friend/2`	**-0.15**	−0.16	**-0.15**	**-0.15**	**-0.15**	**-0.15**	**-0.15**	**-0.15**	**-0.15**
`teaches/2`	**-0.14**	**-0.14**	**-0.14**	**-0.14**	**-0.14**	**-0.14**	**-0.14**	**-0.14**	**-0.14**
Total WPLL	**-18.16**	−21.10	−24.34	−19.27	−22.86	−18.77	−22.67	−19.11	−22.62

Table A.3.2: The per randvar WPLL for each predicate on the synthetic data when training on two interpretations. The best results are in bold.

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
`nrhours/1`	**-4.37**	−5.41	−6.07	−4.82	−5.62	−4.58	−6.07	−4.73	−5.44
`difficulty/1`	**-0.01**	−0.66	−1.31	−0.26	−0.95	−0.06	−0.90	−0.08	−0.85
`ability/1`	**-5.31**	−5.92	−5.92	−5.64	−5.64	−5.65	−5.68	−5.64	−5.68
`intelligence/1`	**-4.85**	−5.71	−6.12	−5.19	−6.08	−4.98	−5.98	−4.89	−5.94
`grade/2`	−1.50	−1.51	−1.53	**-1.45**	−1.53	**-1.45**	−1.53	−1.46	−1.53
`satisfaction/2`	−1.55	−1.55	**-1.54**	−1.55	**-1.54**	−1.55	**-1.54**	−1.55	**-1.54**
`takes/2`	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**
`friend/2`	**-0.15**	**-0.15**	**-0.15**	**-0.15**	**-0.15**	**-0.15**	**-0.15**	**-0.15**	**-0.15**
`teaches/2`	**-0.10**	**-0.14**	**-0.14**	**-0.14**	**-0.10**	**-0.14**	**-0.14**	**-0.14**	**-0.14**
Total WPLL	**-17.89**	−21.06	−20.52	−19.20	−21.65	−18.56	−22.00	−18.64	−21.27

Table A.3.3: The per randvar WPLL for each predicate on the synthetic data when training on four interpretations. The best results are in bold.

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
`nrhours/1`	**-4.35**	−5.41	*OoM*	−4.82	*OoM*	−4.58	*OoM*	−4.73	*OoM*
`difficulty/1`	**-0.02**	−0.67	*OoM*	−0.18	*OoM*	−0.06	*OoM*	−0.09	*OoM*
`ability/1`	**-5.31**	−5.92	*OoM*	−5.63	*OoM*	−5.65	*OoM*	−5.63	*OoM*
`intelligence/1`	**-4.86**	−5.71	*OoM*	−5.18	*OoM*	−4.97	*OoM*	−4.88	*OoM*
`grade/2`	−1.49	−1.51	*OoM*	−1.46	*OoM*	−1.46	*OoM*	**-1.45**	*OoM*
`satisfaction/2`	**-1.55**	**-1.55**	*OoM*	**-1.55**	*OoM*	**-1.55**	*OoM*	**-1.55**	*OoM*
`takes/2`	**-0.00**	**-0.00**	*OoM*	**-0.00**	*OoM*	**-0.00**	*OoM*	**-0.00**	*OoM*
`friend/2`	**-0.15**	**-0.15**	*OoM*	**-0.15**	*OoM*	**-0.15**	*OoM*	**-0.15**	*OoM*
`teaches/2`	−0.14	**-0.14**	*OoM*	**-0.14**	*OoM*	**-0.14**	*OoM*	**-0.14**	*OoM*
Total WPLL	**-17.87**	−21.06	*OoM*	−19.12	*OoM*	−18.55	*OoM*	−18.46	*OoM*

Table A.3.4: The per randvar WPLL for each predicate on the synthetic data when training on eight interpretations. The best results are in bold, and OoM denotes out of memory.

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
`nrhours/1`	**-4.35**	−5.41	*OoM*	−4.79	*OoM*	−4.57	*OoM*	−4.50	*OoM*
`difficulty/1`	**-0.02**	−0.66	*OoM*	−0.15	*OoM*	−0.05	*OoM*	−0.04	*OoM*
`ability/1`	**-5.31**	−5.91	*OoM*	−5.63	*OoM*	−5.64	*OoM*	−5.61	*OoM*
`intelligence/1`	**-4.83**	−5.71	*OoM*	−5.18	*OoM*	−4.96	*OoM*	−4.87	*OoM*
`grade/2`	−1.49	−1.51	*OoM*	−1.46	*OoM*	−1.46	*OoM*	**-1.45**	*OoM*
`satisfaction/2`	**-1.55**	**-1.55**	*OoM*	**-1.55**	*OoM*	**-1.55**	*OoM*	**-1.55**	*OoM*
`takes/2`	**-0.00**	**-0.00**	*OoM*	**-0.00**	*OoM*	**-0.00**	*OoM*	**-0.00**	*OoM*
`friend/2`	**-0.15**	**-0.15**	*OoM*	**-0.15**	*OoM*	**-0.15**	*OoM*	**-0.15**	*OoM*
`teaches/2`	**-0.14**	**-0.14**	*OoM*	**-0.14**	*OoM*	**-0.14**	*OoM*	**-0.14**	*OoM*
Total WPLL	**-17.85**	−21.05	*OoM*	−19.04	*OoM*	−18.52	*OoM*	−18.30	*OoM*

Table A.3.5: The per randvar WPLL for each predicate on the synthetic data when training on 16 interpretations. The best results are in bold, and OoM denotes out of memory.

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
`nrhours/1`	**-4.66**	−5.35	−6.30	−4.94	−6.25	−5.04	−6.11	−5.63	−6.05
`difficulty/1`	**-0.07**	−0.91	−1.59	−0.45	−1.59	−0.28	−1.59	−0.34	−1.59
`ability/1`	−5.35	−5.46	−5.95	−5.21	−5.70	**-5.15**	−5.72	−5.18	−5.67
`intelligence/1`	**-4.73**	−5.54	−7.24	−5.06	−6.09	−4.86	−6.01	−4.93	−5.94
`grade/2`	**-1.51**	**-1.51**	−1.53	**-1.51**	−1.53	**-1.51**	−1.53	−1.52	−1.53
`satisfaction/2`	−1.55	−1.55	**-1.54**	−1.55	**-1.54**	−1.55	**-1.54**	−1.55	**-1.54**
`takes/2`	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**
`friend/2`	−0.16	−0.16	**-0.15**	−0.16	**-0.15**	−0.16	**-0.15**	−0.16	**-0.15**
`teaches/2`	**-0.07**	**-0.07**	−0.14	**-0.07**	−0.14	**-0.07**	−0.14	**-0.07**	−0.14
Total WPLL	**-18.11**	−20.56	−24.45	−18.95	−23.00	−18.62	−22.79	−19.39	−22.62

Table A.3.6: The per randvar WPLL for each predicate on the synthetic data consisting of 100 students, 50 courses and 50 professors. The best results are in bold.

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
`nrhours/1`	**-4.50**	−5.28	−6.11	−4.69	−6.08	−4.70	−6.11	−4.76	−6.13
`difficulty/1`	**-0.05**	−0.71	−1.57	−0.18	−1.57	−0.24	−1.59	−0.14	−1.59
`ability/1`	−5.34	−5.60	−5.60	−5.29	−5.29	**-5.26**	−5.58	**-5.26**	−5.62
`intelligence/1`	**-4.70**	−5.61	−6.00	−5.11	−5.99	−4.77	−6.11	−4.77	−6.14
`grade/2`	−1.48	−1.50	−1.54	**-1.46**	−1.54	−1.47	−1.58	−1.47	−1.59
`satisfaction/2`	−1.55	−1.55	**-1.54**	−1.55	**-1.54**	−1.55	−1.59	−1.55	−1.59
`takes/2`	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	−1.00	**-0.00**	−1.00
`friend/2`	**-0.16**	**-0.16**	**-0.16**	**-0.16**	−0.24	**-0.16**	−1.00	**-0.16**	−1.00
`teaches/2`	**-0.07**	**-0.07**	**-0.07**	**-0.07**	−0.91	−0.08	−1.00	**-0.07**	−1.00
Total WPLL	**-17.84**	−20.47	−22.25	−18.50	−23.15	−18.22	−25.55	−18.17	−25.64

Table A.3.7: The per randvar WPLL for each predicate on the synthetic data consisting of 200 students, 75 courses and 75 professors. The best results are in bold.

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
`nrhours/1`	**-4.52**	−5.32	−6.14	−4.72	−6.09	−4.51	−5.96	−4.76	−5.89
`difficulty/1`	**-0.02**	−0.70	−1.55	−0.17	−1.55	−0.12	−1.55	−0.10	−1.55
`ability/1`	−5.33	−5.53	−5.53	−5.28	−5.28	−5.21	**-5.18**	−5.22	−5.19
`intelligence/1`	**-4.67**	−5.62	−6.01	−5.10	−5.98	−4.79	−5.84	−4.74	−5.78
`grade/2`	−1.46	−1.48	−1.53	**-1.45**	−1.54	**-1.45**	−1.54	−1.46	−1.53
`satisfaction/2`	**-1.54**	**-1.54**	**-1.54**	**-1.54**	**-1.54**	**-1.54**	**-1.54**	**-1.54**	**-1.54**
`takes/2`	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**
`friend/2`	**-0.16**	**-0.16**	**-0.16**	**-0.16**	**-0.16**	**-0.16**	**-0.16**	**-0.16**	**-0.16**
`teaches/2`	**-0.07**	**-0.07**	**-0.07**	**-0.07**	**-0.07**	**-0.07**	**-0.07**	**-0.07**	**-0.07**
Total WPLL	**-17.78**	−20.42	−22.53	−18.48	−22.20	−17.86	−21.83	−18.05	−21.71

Table A.3.8: The per randvar WPLL for each predicate on the synthetic data consisting of 400 students, 100 courses and 100 professors. The best results are in bold.

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
`nrhours/1`	**-4.48**	−5.33	−6.18	−4.78	−5.96	−4.54	−5.99	−4.58	−5.92
`difficulty/1`	**-0.02**	−0.66	−1.53	−0.17	−1.55	−0.09	−1.53	−0.12	−1.53
`ability/1`	**-5.34**	−5.67	−5.67	−5.31	−5.18	−5.26	−5.26	−5.24	−5.24
`intelligence/1`	**-4.66**	−5.65	−6.04	−5.13	−5.84	−4.79	−5.84	−4.73	−5.79
`grade/2`	−1.45	−1.47	−1.54	**-1.44**	−1.54	**-1.44**	−1.53	**-1.44**	−1.53
`satisfaction/2`	−1.54	−1.54	−1.54	−1.54	−1.54	**-1.53**	−1.54	**-1.53**	−1.54
`takes/2`	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**	**-0.00**
`friend/2`	**-0.15**	**-0.15**	**-0.15**	−0.15	−0.16	**-0.15**	**-0.15**	−0.16	**-0.15**
`teaches/2`	**-0.07**	**-0.07**	**-0.07**	**-0.07**	**-0.07**	**-0.07**	**-0.07**	**-0.07**	**-0.07**
Total WPLL	**-17.72**	−20.54	−22.72	−18.60	−21.83	−17.87	−21.92	−17.86	−21.79

Table A.3.9: The per randvar WPLL for each predicate on the synthetic data consisting of 800 students, 125 courses and 125 professors. The best results are in bold.

	Hybrid	Discretized into 2 bins		Discretized into 4 bins		Discretized into 6 bins		Discretized into 8 bins	
Predicate	HRDN	HRDN	LSM	HRDN	LSM	HRDN	LSM	HRDN	LSM
`avgNrWith/1`	−4.60	−4.81	−4.83	−4.61	−4.75	**-4.56**	−4.85	−4.60	−5.05
`avgSalary/1`	−11.24	−11.35	−11.44	−10.96	−11.25	−10.82	−11.39	**-10.68**	−11.65
`avgSumofCred/1`	**-14.54**	−17.31	−17.58	−16.44	−17.11	−16.01	−16.34	−15.70	−16.88
`avgSumOfW/1`	**-14.39**	−17.22	−17.43	−16.34	−17.00	−15.85	−16.26	−15.54	−16.81
`clientAge/2`	−5.77	−5.72	−5.71	−5.67	−5.70	−5.60	−5.74	**-5.59**	−5.73
`clientDistrict/2`	**-0.09**	**-0.09**	−0.11	**-0.09**	**-0.09**	**-0.09**	−0.12	**-0.09**	−0.15
`freq/1`	**-0.38**	−0.41	−0.45	−0.39	−0.42	**-0.38**	−0.50	−0.39	−0.45
`gender/1`	**-1.00**	**-1.00**	**-1.00**	**-1.00**	**-1.00**	**-1.00**	**-1.00**	**-1.00**	**-1.00**
`hasAccount/2`	**-0.02**	**-0.02**	−0.03	**-0.02**	**-0.02**	**-0.02**	−0.03	**-0.02**	−0.03
`hasLoan/2`	**-0.01**	**-0.01**	−0.03	−0.07	−0.04	−0.06	−0.03	−0.13	−0.03
`loanAmount/1`	−18.26	−18.48	−18.54	**-18.17**	−18.40	−18.19	−18.62	−18.35	−18.80
`loanStatus/1`	**-1.32**	−1.42	−1.48	−1.40	−1.43	−1.40	−1.45	−1.50	−1.48
`monthlyPayments/1`	−12.70	−12.96	−13.02	**-12.69**	−12.89	−12.91	−13.09	−12.80	−13.15
`ratUrbInhab/1`	−5.74	−5.83	−5.93	−5.56	−5.77	−5.40	−5.86	**-5.29**	−5.91
`stdOfCred/1`	**-14.05**	−15.65	−15.90	−14.98	−15.51	−14.67	−15.51	−14.34	−16.10
`stdOfW/1`	**-13.81**	−15.36	−15.56	−14.62	−15.18	−14.32	−15.06	−14.16	−15.66
Total WPLL	**-117.93**	−127.64	−129.02	−122.99	−126.55	−121.27	−125.83	−120.17	−128.89

Table A.3.10: The per randvar WPLL for each predicate on the PKDD’99 financial data set. The best results are in bold.

Predicate	LR+LinR	LR+MP5	LLM-H
`avgNrWith/1`	−0.96	−0.96	**-0.09**
`avgSalary/1`	**-1.00**	**-1.00**	**-1.00**
`avgSumofCred/1`	−1.00	−1.00	**-0.02**
`avgSumOfW/1`	**-0.36**	**-0.36**	−0.38
`clientAge/2`	−0.89	−0.89	**-0.01**
`clientDistrict/2`	**-1.17**	**-1.17**	−1.32
`freq/1`	−5.78	**-5.77**	**-5.77**
`gender/1`	−11.18	**-10.98**	−11.24
`hasAccount/2`	**-5.74**	**-5.74**	**-5.74**
`hasLoan/2`	**-14.35**	−14.98	−14.39
`loanAmount/1`	**-14.51**	−15.08	−14.54
`loanStatus/1`	**-13.81**	−13.82	**-13.81**
`monthlyPayments/1`	**-14.05**	−13.89	**-14.05**
`ratUrbInhab/1`	−4.31	**-4.10**	−4.60
`stdOfCred/1`	**-18.14**	−18.15	−18.26
`stdOfW/1`	**-12.54**	−12.61	−12.70
Total WPLL	−119.79	−120.22	**-117.93**

Table A.3.11: The per randvar WPLL of the two variants of the handcrafted models, LR+LinR and LR+MP5, compared to LLM-H on the hybrid data for the PKDD'99 financial data set. LR+LinR uses logistic regression for discrete predicates and linear regression for continuous predicates, and LR+MP5 uses logistic regression for discrete predicates and regression trees for continuous predicates. The best results are in bold.

A.4 Features used for propositional learners

In order to compare our structure learning algorithm to propositional learners on the PKDD'99 financial data set, we handcrafted a number of features for each of the 16 predicates. Each feature predicts a property of an object by using some other properties of that object.

A.4.1 Predicates with discrete range

clientDistrict(C,D)	$\mathit{value}_{\{C\}}$(gender(C),$\emptyset$) value$_{\{C\}}$(avgSalary(D),$\emptyset$) value$_{\{C\}}$(ratUrbInhab(D),$\emptyset$)

gender(C)	$\mathit{exists}_{\{C\}}$(hasAccount(A,C),$\emptyset$)

hasAccount(C,A)	value$_{\{C\}}$(gender(C),$\emptyset$) $\mathit{exists}_{\{A\}}$($\emptyset$,hasLoan(A,L)) value$_{\{A\}}$(freq(A),$\emptyset$) value$_{\{A\}}$(avgNrWith(A,$\emptyset$) value$_{\{A\}}$(avgSumOfW(A),$\emptyset$) value$_{\{A\}}$(avgSumOfCred(A),$\emptyset$) value$_{\{A\}}$(stdOfW(A),$\emptyset$) value$_{\{A\}}$(stdOfCred(A),$\emptyset$)

freq(A)	value$_{\{A\}}$(avgNrWith(A),$\emptyset$) value$_{\{A\}}$(avgSumOfW(A),$\emptyset$) value$_{\{A\}}$(avgSumOfCred(A),$\emptyset$) value$_{\{A\}}$(stdOfW(A),$\emptyset$) value$_{\{A\}}$(stdOfCred(A),$\emptyset$)

hasLoan(A,L)	$\text{value}_{\{L\}}$(loanAmount(L),$\emptyset$) $\text{value}_{\{L\}}$(loanStatus(L),$\emptyset$) $\text{value}_{\{L\}}$(monthlyPayments(L),$\emptyset$) $\text{value}_{\{A\}}$(freq(A),$\emptyset$) $\text{value}_{\{A\}}$(avgNrWith(A),$\emptyset$) $\text{value}_{\{A\}}$(avgSumOfW(A),$\emptyset$) $\text{value}_{\{A\}}$(avgSumOfCred(A),$\emptyset$) $\text{value}_{\{A\}}$(stdOfW(A),$\emptyset$) $\text{value}_{\{A\}}$(stdOfCred(A),$\emptyset$)

loanStatus(L)	$\text{value}_{\{L\}}$(loanAmount(L),$\emptyset$) $\text{value}_{\{L\}}$(loanStatus(L),$\emptyset$) $\text{value}_{\{L\}}$(monthlyPayments(L),$\emptyset$)

A.4.2 Predicates with continuous range

The following are the features we used for predicates with a continuous range. Note that the predicates avgSumOfW/1, avgSumOfCred/1, stdOfW/1 and stdOfCred/1 have similar structure as the features for avgNrWith(A). To save space we will only show the features we used for avgNrWith(A). The full feature set is in the online appendix on http://dtai.cs.kuleuven.be/ml/systems/llm.

avgSalary(D)	$\textit{proportion}_{\{D\}}$(clientDistrict(C,D),$\emptyset$) $\text{value}_{\{D\}}$(ratUrbInhab(D),$\emptyset$)

loanAmount(L)	$\texttt{value}_{\{L\}}(\texttt{loanStatus(L)},\emptyset)$ $\texttt{value}_{\{L\}}(\texttt{monthlyPayments(L)},\emptyset)$

monthlyPayments(L)	$\texttt{value}_{\{L\}}(\texttt{loanStatus(L)},\emptyset)$ $\texttt{value}_{\{L\}}(\texttt{loanAmount(L)},\emptyset)$

avgNrWith(A)	$\texttt{value}_{\{A\}}(\texttt{freq(A)},\emptyset)$ $\texttt{value}_{\{A\}}(\texttt{avgSumOfW(A)},\emptyset)$ $\texttt{value}_{\{A\}}(\texttt{avgSumOfCred(A)},\emptyset)$ $\texttt{value}_{\{A\}}(\texttt{stdOfW(A)},\emptyset)$ $\texttt{value}_{\{A\}}(\texttt{stdOfCred(A)},\emptyset)$

ratUrbInhab(D)	$\texttt{value}_{\{D\}}(\texttt{avgSalary(D)},\emptyset)$ $\textit{proportion}_{\{D\}}(\texttt{clientDistrict(C,D)},\emptyset)$

clientAge(C,L)	$\texttt{value}_{\{C\}}(\texttt{gender(C)},\emptyset)$ $\texttt{value}_{\{L\}}(\texttt{loanAmount(L)},\emptyset)$ $\texttt{value}_{\{L\}}(\texttt{loanStatus(L)},\emptyset)$ $\texttt{value}_{\{L\}}(\texttt{monthlyPayments(L)},\emptyset)$

Bibliography

Ahmadi, B., K. Kersting, and S. Natarajan (2012). “Lifted online training of relational models with stochastic gradient methods”. In: *Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 585–600 (p. 136).

Ariely, D. (2008). *Predictably irrational: The hidden forces that shape our decisions*. Harper Collins (p. 61).

Arnold, B. C. and S. J. Press (1989). “Compatible conditional distributions”. In: *Journal of the American Statistical Association* 84(405), pp. 152–156 (p. 18).

Atkeson, C. G. and S. Schaal (1997). “Robot learning from demonstration”. In: *ICML*. Vol. 97, pp. 12–20.

Barabasi, A.-L. and R. Albert (1999). “Emergence of scaling in random networks”. In: *Science* 286(5439), pp. 509–512 (p. 59).

Baskerville, K., P. Grassberger, and M. Paczuski (2007). “Graph animals, subgraph sampling, and motif search in large networks”. In: *Physical Review E* 76(3 Pt 2), p. 036107 (p. 71).

Berka, P. (1999). PKDD’99 Discovery challenge: `http://lisp.vse.cz/pkdd99/Challenge/` (p. 96).

Besag, J. (1974). “Spatial interaction and the statistical analysis of lattice systems”. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 36, pp. 192–236 (p. 18).

Bishop, C. M. (1995). *Neural networks for pattern recognition*. New York, NY, USA: Oxford University Press, Inc.

Bishop, C. M. (2006). "Pattern recognition and machine learning". In: *Machine Learning* 128 (pp. 15, 16, 80, 92, 122).

Blockeel, H. and L. De Raedt (1998). "Top-down induction of first-order logical decision trees". In: *Artificial intelligence* 101, pp. 285–297 (pp. 36, 108, 114, 132).

Bordino, I., D. Donato, A. Gionis, and S. Leonardi (2008). "Mining large networks with subgraph counting". In: *Proceedings of the Eighth IEEE International Conference on Data Mining*. (ICDM), (2008). Washington, DC, USA, pp. 737–742 (pp. 71, 72).

Boutilier, C., N. Friedman, M. Goldszmidt, and D. Koller (1996). "Context-specific independence in Bayesian networks". In: *Proceedings of the 12th international conference on Uncertainty in artificial intelligence (UAI), (1996)*. Morgan Kaufmann Publishers Inc. (p. 20).

Bratko, I. (2001). *Prolog programming for artificial intelligence*. Pearson education (p. 28).

Braz, R., E. Amir, and D. Roth (2008). "A survey of first-order probabilistic models". In: *Innovations in Bayesian Networks*. Vol. 156. Studies in Computational Intelligence. Springer Berlin Heidelberg, pp. 289–317.

Choi, J., E. Amir, and D. J. Hill (2010). "Lifted inference for relational continuous models". In: *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence (UAI), (2010)*, pp. 126–134 (p. 83).

Clark, P. and R. Boswell (1991). "Rule induction with CN2: some recent improvements". In: *Machine learning-EWSL-91*. Springer, pp. 151–163 (p. 12).

Cobb, B., R. Rumí, and A. Salmerón (2007). "Bayesian network models with discrete and continuous variables". In: *Advances in probabilistic graphical models*. Vol. 214. Springer Berlin Heidelberg, pp. 81–102 (p. 107).

Codd, E. F. (1970). "A relational model of data for large shared data banks". In: *Communications of the ACM* 13(6), pp. 377–387.

Cordella, L., P. Foggia, C. Sansone, and M. Vento (2004). "A (sub)graph isomorphism algorithm for matching large graphs". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 26(10), pp. 1367–1372 (p. 71).

Croonenborghs, T., J. Ramon, H. Blockeel, and M. Bruynooghe (2007). "Online learning and exploiting relational models in reinforcement learning". In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), (2007)*. Vol. 2007, pp. 726–731.

Das, M., Y. Wu, T. Khot, K. Kersting, and S. Natarajan (2016). "Scaling lifted probabilistic inference and learning via graph databases". In: *Proceedings of* *the 2016 SIAM International Conference on Data Mining*. Chap. 83, pp. 738–746. (Pp. 43, 72).

Date, C. J. and H. Darwen (1997). *A guide to SQL standard*. Vol. 3. Addison-Wesley Reading (p. 27).

Davis, J. and P. Domingos (2009). "Deep transfer via second-order Markov logic". In: *Proceedings of the 26th International Conference on Machine Learning (ICML), (2009)*, pp. 217–224 (p. 60).

Davis, J., E. Burnside, I. de Castro Dutra, D. Page, R. Ramakrishnan, V. S. Costa, and J. W. Shavlik (2005). "View learning for statistical relational learning: with an application to mammography." In: *Proceedings of the 19th International Joint Conference of Artificial Intelligence (IJCAI), (2005)*. Citeseer, pp. 677–683 (p. 24).

Davis, J., E. Burnside, D. Page, and V. S. Costa (2006). "View learning extended: inventing new tables for statistical relational learning". In: *Proceedings of Open Problems in Statistical Relational Learning Workshop* (p. 138).

De Raedt, L. (2008). *Logical and relational learning: from ILP to MRDM (Cognitive Technologies)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc. (P. 2).

De Raedt, L., P. Frasconi, K. Kersting, and S. Muggleton, eds. (2008). *Probabilistic inductive logic programming — Theory and applications*. Vol. 4911. LNAI. Springer (p. 2).

De Raedt, L. (1998). "Attribute-value learning versus inductive logic programming: The missing links". In: *Lecture notes in computer science vol:1446 pages:1-8*. Springer, pp. 1–8.

De Raedt, L. and K. Kersting (2008). *Probabilistic inductive logic programming*. Springer (p. 84).

De Raedt, L. and K. Kersting (2011). "Statistical relational learning". In: *Encyclopedia of Machine Learning*. Springer, pp. 916–924 (pp. 2, 29).

De Raedt, L., K. Kersting, S. Natarajan, and D. Poole (2016). "Statistical relational artificial intelligence: logic, probability, and computation". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 10(2), pp. 1–189 (p. 29).

De Raedt, L., A. Kimmig, and H. Toivonen (2007). "ProbLog: a probabilistic prolog and its application in link discovery". In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI), (2007)*. Vol. 7, pp. 2462–2467 (p. 29).

Depiero, F., M. Trivedi, and S. Serbin (1996). "Graph matching using a direct classification of node attendance". In: *Pattern Recognition* 29(6), pp. 1031–1048 (p. 137).

Di Natale, R., A. Ferro, R. Giugno, M. Mongiovi, A. Pulvirenti, and D. Shasha (2010). "SING: subgraph search in non-homogeneous graphs". In: *BMC Bioinformatics* 11(1), p. 96 (p. 71).

Dobra, A. (2009). "Variable selection and dependency networks for genomewide data". In: *Biostatistics (Oxford, England)* 10, pp. 621–639 (pp. 82, 107, 110).

Domingos, P. and M. Richardson (2004). "Markov logic: a unifying framework for statistical relational learning". In: *Proocedings of the 21st International Conference on Machine Learning (ICML-2004) worskhop on statistical relational learning and its connections to other fields*, pp. 49–54 (p. 5).

Fierens, D., H. Blockeel, M. Bruynooghe, and J. Ramon (2005). "Logical Bayesian networks and their relation to other probabilistic logical models". In: *Proceedings of the 15th International Conference on Inductive Logic Programming (ICLP), (2005)*. Vol. 3625. Springer, pp. 121–135 (pp. 4, 29, 30, 41, 83, 84).

Fierens, D., H. Blockeel, J. Ramon, and M. Bruynooghe (2004). "Logical Bayesian networks". In: *Proceedings of the 3nd international workshop on multi-relational data mining (2004)*, pp. 19–30 (p. 63).

Friedman, N. (1997). "Learning belief networks in the presence of missing values and hidden variables". In: *Proceedings of the 14th International Conference on Machine Learning (ICML), (1997)*. Vol. 97, pp. 125–133 (p. 137).

Friedman, N. (1998). "The Bayesian structural EM algorithm". In: *Proceedings of the 14th conference on Uncertainty in artificial intelligence (UAI), (1998)*. Morgan Kaufmann Publishers Inc., pp. 129–138.

Friedman, N. and M. Goldszmidt (1996). "Learning Bayesian networks with local structure". In: *Proceedings of the 12th International Conference on Uncertainty in Artificial Intelligence (UAI), (1996)*. UAI'96. Portland, OR, pp. 252–262. (P. 64).

Fürer, M. and P. S. Kasiviswanathan (2008). "Approximately counting embeddings into random graphs". In: *Proceedings of the 11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008 on Approximation, Randomization and Combinatorial Optimization: Algorithms and Techniques*. APPROX '08 / RANDOM '08. Boston, MA, USA, pp. 416–429. (Pp. 3, 4, 40, 41, 47, 51, 53, 58, 72, 134).

Gardiol, N. H. and L. P. Kaelbling (2007). "Action-space partitioning for planning". In: *In Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI), (2007)*. Vol. 22. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, p. 980 (p. 138).

Genesereth, M. (2010). "Data integration: the relational logic approach". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 4(1), pp. 1–97.

Genesereth, M. and N. Nilsson (1987). "Logical foundations of artificial intelligence". In: *Intelligence. Morgan Kaufmann* (p. 84).

Getoor, L., N. Friedman, D. Koller, and A. Pfeffer (2001). "Learning probabilistic relational models". In: *Relational Data Mining*. Springer-Verlag (pp. 21, 23, 30, 83, 95).

Getoor, L. and B. Taskar (2007). *Introduction to statistical relational learning*. The MIT press (pp. 2, 29).

Giugno, R. and D. Shasha (2002). "GraphGrep: A fast and universal method for querying graphs". In: *Proceedings of the 16th International Conference on Pattern Recognition (2002)*. Vol. 2, 112–115 vol.2 (p. 71).

Guo, Y. and S. Gu (2011). "Multi-label classification using conditional dependency networks". In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI), (2011)*. IJCAI'11. Barcelona, Catalonia, Spain, pp. 1300–1305 (p. 108).

Gutmann, B., M. Jaeger, and L. De Raedt (2011). "Extending ProbLog with continuous distributions". In: *Proceedings of the 20th International Conference on Inductive Logic Programming (ILP), (2011)*. ILP'10. Florence, Italy, pp. 76–91. (Pp. 83, 108).

Gutmann, B., I. Thon, A. Kimmig, M. Bruynooghe, and L. De Raedt (2011). "The magic of logical inference in probabilistic programming". In: *Theory and Practice of Logic Programming* 11(4–5), pp. 663–680 (pp. 38, 114).

Hadiji, F., A. Molina, S. Natarajan, and K. Kersting (2015). "Poisson dependency networks: gradient boosted models for multivariate count data". In: *Machine Learning* 100(2-3), pp. 477–507 (p. 83).

Hall, M., E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten (2009). "The WEKA data mining software: an update". In: *SIGKDD Explorations* 11(1), pp. 10–18 (p. 97).

Heckerman, D. (1999). "A tutorial on learning with Bayesian networks". In: *Learning in Graphical Models*. Cambridge, MA, USA: MIT Press, pp. 301–354 (p. 21).

Heckerman, D., D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie (2001). "Dependency networks for inference, collaborative filtering, and data visualization". In: *The Journal of Machine Learning Research* 1, pp. 49–75 (pp. 5, 12, 15, 17, 22).

Hulten, G., D. M. Chickering, and D. Heckerman (2003). "Learning Bayesian networks from dependency networks: a preliminary study". In: *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics (AISTATS), (2003)* (p. 136).

Inokuchi, A., T. Washio, and H. Motoda (2003). "Complete mining of frequent patterns from graphs: mining graph data". In: *Machine Learning* 50(3), pp. 321–354 (p. 71).

Jaeger, M. (1997). "Relational Bayesian networks". In: *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*. UAI'97. Providence, Rhode Island, pp. 266–273.

Jolad, S., A. Roman, and M. C. Shastry (2012). "A bounded divergence measure based on the Bhattacharyya coefficient". In: *ArXiv e-prints*. arXiv: 1201.0418 [math.ST].

Joshi, S., R. Khardon, P. Tadepalli, A. Fern, and A. Raghavan (2013). "Relational Markov decision processes: promise and prospects". In: *Proceedings of the 16th AAAI Conference on Statistical Relational Artificial Intelligence*. AAAIWS'13-16, pp. 15–17.

Jowhari, H. and M. Ghodsi (2005). "New streaming algorithms for counting triangles in graphs". In: *COCOON-05*, pp. 710–716 (p. 50).

Kashtan, N., S. Itzkovitz, R. Milo, and U. Alon (2004). "Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs". In: *Bioinformatics* 20(11), pp. 1746–1758 (p. 71).

Kersting, K. and L. De Raedt (2001). "Adaptive Bayesian logic programs". In: *Proceedings of the 11th International Conference on Inductive Logic Programming (ILP), (2001)*. Springer, pp. 104–117 (pp. 83, 108).

Kersting, K. and L. De Raedt (2007). "Bayesian logic programming: theory and tool". In: *Statistical Relational Learning*, p. 291.

Kersting, K., L. De Raedt, and S. Kramer (2000). "Interpreting Bayesian logic programs". In: *Proceedings of the AAAI-2000 workshop on learning statistical models from relational data*, pp. 29–35 (pp. 29, 63).

Kersting, K. and K. Driessens (2008). "Non-parametric policy gradients: a unified treatment of propositional and relational domains". In: *Proceedings of the 25th International Conference on Machine Learning (ICML), (2008)*. Helsinki, Finland, pp. 456–463 (p. 131).

Kersting, K. and L. D. Raedt (2001). "Towards combining inductive logic programming with Bayesian networks". In: *Proceedings of the 11th International Conference on Inductive Logic Programming (ILP), (2001)*. ILP '01. London, UK, UK, pp. 118–131.

Khosravi, H. and B. Bina (2010). "A survey on statistical relational learning". In: *Proceedings of the 23rd Canadian conference on Advances in Artificial Intelligence*. AI'10. Ottawa, Canada, pp. 256–268.

Khot, T., S. Natarajan, K. Kersting, and J. Shavlik (2012). "Structure learning with hidden data in relational domains". In: (p. 137).

Khot, T., S. Natarajan, K. Kersting, and J. Shavlik (2015). “Gradient-based boosting for statistical relational learning: the Markov logic network and missing data cases”. In: *Machine Learning* 100(1), pp. 75–100 (p. 135).

King, R. D., A. Srinivasan, and L. Dehaspe (2001). “WARMR: a data mining tool for chemical data”. In: *Journal of Computer-Aided Molecular Design* 15(2), pp. 173–181 (p. 22).

Kok, S. and P. Domingos (2005). “Learning the structure of Markov logic networks”. In: *Proceedings of the 22Nd International Conference on Machine Learning (ICML), (2005)*. ICML ’05, pp. 441–448 (pp. 72, 98).

Kok, S. and P. Domingos (2010). “Learning Markov logic networks using structural motifs”. In: *Proceedings of the 27th International Conference on Machine Learning (ICML), (2010)*, pp. 551–558 (p. 97).

Koller, D. (1999). “Probabilistic relational models”. In: *Inductive Logic Programming, 9th International Workshop, ILP-99*, pp. 3–13.

Koller, D. and N. Friedman (2009). *Probabilistic graphical models: principles and techniques*. MIT press (pp. 12, 14).

Koller, D., U. Lerner, and D. Angelov (1999). “A general algorithm for approximate inference and its application to hybrid Bayes nets”. In: *Proceedings of the 15th conference on Uncertainty in artificial intelligence (UAI), (1999)*. Morgan Kaufmann Publishers Inc., pp. 324–333 (pp. 81, 107, 137).

Kramer, S., N. Lavrač, and P. Flach (2000). “Relational data mining”. In: New York, NY, USA: Springer-Verlag New York, Inc. Chap. Propositionalization Approaches to Relational Data Mining, pp. 262–286. (P. 23).

Kuželka, O., A. Szabóová, M. Holec, and F. Železný (2011). “Gaussian logic for predictive classification”. In: *Machine Learning and Knowledge Discovery in Databases*. Vol. 6912. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 277–292 (p. 83).

Landwehr, N., K. Kersting, and L. De Raedt (2005). “nFOIL: integrating naıve Bayes and FOIL”. In: *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pp. 795–800 (p. 24).

Landwehr, N., K. Kersting, and L. D. Raedt (2007). “Integrating naıve Bayes and FOIL”. In: *Journal of Machine Learning Research* 8(Mar), pp. 481–507 (p. 24).

Lang, T. and M. Toussaint (2010). “Planning with noisy probabilistic relational rules”. In: *Journal of Artificial Intelligence Research* 39, pp. 1–49 (p. 131).

Lang, T., M. Toussaint, and K. Kersting (2012). “Exploration in relational domains for model-based reinforcement learning”. In: *Journal of Machine Learning Research* 13(Dec), pp. 3725–3768 (p. 37).

Lauritzen, S. L. (1992). “Propagation of probabilities, means and variances in mixed graphical association models”. In: *Journal of the American Statistical Association* 87, pp. 1098–1108 (p. 81).

Lauritzen, S. L. and F. Jensen (2001). “Stable local computation with conditional gaussian distributions”. In: *Statistics and Computing* 11, pp. 191–203 (p. 107).

Lauritzen, S. and N. Wermuth (1989). “Graphical models for associations between variables, some of which are qualitative and some quantitative”. In: *The Annals of Statistics*, pp. 31–57 (p. 82).

Lerner, U., E. Segal, and D. Koller (2001). “Exact inference in networks with discrete children of continuous parents”. In: *Proceedings of the 7th conference on Uncertainty in artificial intelligence (UAI), (2001)*. Morgan Kaufmann Publishers Inc., pp. 319–328 (p. 83).

Leskovec, J. and C. Faloutsos (2006). “Sampling from large graphs”. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD ’06. New York, NY, USA, pp. 631–636 (p. 71).

Lloyd, J. W. (2012). *Foundations of logic programming*. Springer Science & Business Media (p. 25).

Lowd, D. (2012). “Closed-form learning of Markov networks from dependency networks”. In: *arXiv preprint arXiv:1210.4896* (pp. 18, 136).

Lowd, D. and A. Shamaei (2011). “Mean field inference in dependency networks: an empirical study”. In: *Proceedings of the 25th Conference on Artificial Intelligence (AAAI), (2011)*. AAAI’11. San Francisco, California, pp. 404–410. (P. 137).

Luo, B. and E. R. Hancock (2001). “Structural graph matching using the EM algorithm and singular value decomposition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23(10), pp. 1120–1136 (p. 137).

Meinshausen, N. and P. Bühlmann (2006). “High dimensional graphs and variable selection with the lasso”. In: *Annals of statistics* 34, pp. 1436–1462 (pp. 107, 110).

Mewes, H. W., D. Frishman, C. Gruber, B. Geier, D. Haase, A. Kaps, K. Lemcke, G. Mannhaupt, F. Pfeiffer, C. Schüller, S. Stocker, and B. Weil (2000). “MIPS: a database for genomes and protein sequences”. In: *Nucleic Acids Research* 28(1), pp. 37–40. (P. 60).

Mitchell, T. (1997). *Machine Learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc. (Pp. 12, 92).

Moldovan, B. and L. De Raedt (2014). “Learning relational affordance models for two-arm robots”. In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, pp. 2916–2922 (p. 131).

Moldovan, B., P. Moreno, M. van Otterlo, J. Santos-Victor, and L. De Raedt (2012). "Learning relational affordance models for robots in multi-object manipulation tasks". In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, pp. 4373–4378 (p. 131).

Moral, S., R. Rumi, and A. Salmerón (2001). "Mixtures of truncated exponentials in hybrid Bayesian networks". In: *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*. Vol. 2143. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 156–167 (p. 107).

Muggleton, S. (1991). "Inductive logic programming". In: *New generation computing* 8(4), pp. 295–318.

Muggleton, S. and L. De Raedt (1994). "Inductive logic programming: theory and methods". In: *The Journal of Logic Programming* 20, pp. 629–679 (pp. 24, 36).

Munzer, T., B. PIOT, M. Geist, O. Pietquin, and M. Lopes (2015). "Inverse reinforcement learning in relational domains". In: *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI), (2015)*.

Murphy, K. P. (1998). *Inference and learning in hybrid Bayesian networks*. Tech.Rept. UCB/CSD-98-990, U.C. Berkeley, CA (pp. 82, 107).

Narman, P., M. Buschle, J. Konig, and P. Johnson (2010). "Hybrid probabilistic relational models for system quality analysis". In: *Proceedings of the 14th IEEE International Enterprise Distributed Object Computing Conference, (2010)*, pp. 57–66 (pp. 83, 108).

Natarajan, S., S. Joshi, P. Tadepalli, K. Kersting, and J. Shavlik (2011). "Imitation learning in relational domains: a functional-gradient boosting approach". In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI), (2011)*. Vol. 22. 1, p. 1414 (p. 131).

Natarajan, S., T. Khot, K. Kersting, B. Gutmann, and J. Shavlik (2012). "Gradient-based boosting for statistical relational learning: the relational dependency network case". In: *Machine Learning* 86, pp. 25–56 (pp. 98, 108, 135).

Neapolitan, R. E. (2003). "Learning Bayesian networks". In: (p. 10).

Neville, J., D. Jensen, L. Friedland, and M. Hay (2003). "Learning relational probability trees". In: *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 625–630 (pp. 37, 108).

Neville, J., M. Rattigan, and D. Jensen (2003). "Statistical relational learning: four claims and a survey". In: *Proceedings of the Workshop on Learning Statistical Models from Relational Data, Eighteenth International Joint Conference on Artificial Intelligence*.

Neville, J. and D. Jensen (2007). “Relational dependency networks”. In: *The Journal of Machine Learning Research* 8, pp. 653–692 (pp. 5, 29, 35, 36, 43, 78, 83, 108).

Ngo, V. and M. Toussaint (2014). “Model-based relational reinforcement learning when object existence is partially observable”. In: *Proceedings of the 31st International Conference on Machine Learning (ICML), (2014)*, pp. 559–567.

Nitti, D., V. Belle, and L. De Raedt (2015). “Planning in discrete and continuous Markov decision processes by probabilistic programming”. In: *Machine Learning and Knowledge Discovery in Databases*. Springer International Publishing, pp. 327–342 (pp. 6, 39).

Nitti, D., T. De Laet, and L. De Raedt (2013). “A particle filter for hybrid relational domains”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan*, pp. 2764–2771 (pp. 38, 39).

Nitti, D., I. Ravkic, J. Davis, and L. De Raedt (2016). “Learning the structure of dynamic hybrid relational models”. In: *22nd European Conference on Artificial Intelligence (ECAI) 2016, European Conference on Artificial Intelligence (ECAI), The Hague, The Netherlands, 29 - 2 September 2016*, pp. 1283–1290.

Pasula, H., L. S. Zettlemoyer, and L. P. Kaelbling (2004). “Learning probabilistic relational planning rules”. In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 73–82 (p. 131).

Pasula, H., L. Zettlemoyer, and L. P. Kaelbling (2007). “Learning symbolic models of stochastic domains”. In: *Journal of Artificial Intelligence Research*, pp. 309–352 (p. 131).

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann (p. 15).

Plotkin, G. D. (1970). “A note on inductive generalization”. In: *Machine intelligence* 5(1), pp. 153–163 (p. 36).

Poole, D. and N. L. Zhang (2003). “Exploiting contextual independence in probabilistic inference”. In: *Journal of Artificial Intelligence Research (JAIR)* 18, pp. 263–313 (p. 136).

Popescul, A. and L. H. Ungar (2003). “Statistical relational learning for link prediction”. In: *International Joint Conference of 18th Artificial Intelligence (IJCAI) workshop on learning statistical models from relational data*. Vol. 2003. Citeseer (p. 28).

Prettejohn, B. J., M. J. Berryman, and M. D. McDonnell (2011). “Methods for generating complex networks with selected structural properties for simulations: a review and tutorial for neuroscientists”. In: *Frontiers in Computational Neuroscience* 5(11).

Provost, F. and P. Domingos (2000). *Well-trained PETs: Improving probability estimation trees*. CeDER Working Paper #IS-00-04, Stern School of Business, New York University, NY, NY 10012. (p. 99).

Pržulj, N. (2007). "Biological network comparison using graphlet degree distribution". In: *Bioinformatics* 23(2), e177–e183 (p. 71).

Quinlan, J. R. (1986). "Induction of decision trees". In: *Machine learning* 1(1), pp. 81–106 (pp. 12, 13).

Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.

Ravkic, I., J. Ramon, and J. Davis (2012). "Hybrid logical Bayesian networks". In: *Late Breaking Papers proceedings of the 22nd International Conference on Inductive Logic Programming 62-67* (p. 4).

Ravkic, I., J. Ramon, and J. Davis (2015). "Learning relational dependency networks in hybrid domains". In: *Machine Learning* 100(2), pp. 217–254. (Pp. 6, 114, 131).

Ravkic, I., B. Wittevrongel, W. Meert, J. Davis, T. A. Gerbrands, and B. Vanwanseele (2015). "Predicting gait retraining strategies for knee osteoarthritis". In: Workshop on Machine Learning in Life Sciences, ECML-PKDD 2015, Porto, Portugal (p. 12).

Ravkic, I., M. Znidarsic, J. Ramon, and J. Davis (2016). "Graph sampling for efficient parameter estimation in statistical relational learning". In: *(under revision) ECML-PKDD Data mining and Knowledge discovery journal*.

Richards, B. L. and R. J. Mooney (1992). "Learning relations by pathfinding". In: *Proceedings of the 10th national conference on Artificial intelligence (AAAI), (1992)*. AAAI'92, pp. 50–55 (p. 59).

Richardson, M. and P. Domingos (2006). "Markov logic networks". In: *Machine learning* 62, pp. 107–136 (pp. 29, 30, 63, 84, 95).

Romero, V., R. Rumí, and A. Salmerón (2006). "Learning hybrid Bayesian networks using mixtures of truncated exponentials". In: *International Journal of Approximate Reasoning* 42, pp. 54–68 (p. 107).

Roth, D. and W.-t. Yih (2001). "Relational learning via propositional algorithms: an information extraction case study". In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI), (2001)*. Vol. 17. 1, pp. 1257–1263 (p. 24).

Sato, T. and Y. Kameya (1997). "PRISM: a language for symbolic-statistical modeling". In: *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI), (1997)*. Vol. 97, pp. 1330–1339 (p. 29).

Schwarz, G. (1978). "Estimating the dimension of a model". In: *The annals of statistics*, pp. 461–464 (pp. 21, 92).

Shervashidze, N., S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt (2009). “Efficient graphlet kernels for large graph comparison”. In: *12th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Clearwater Beach, Fl, USA, April, 16-18, 2009 (p. 71).

Silverman, B. W. (1986). *Density estimation for statistics and data analysis*. Vol. 26. CRC press.

Sutton, R. S. and A. G. Barto (1998). *Reinforcement learning: An introduction*. MIT press.

Tadepalli, P., R. Givan, and K. Driessens (2004). “Relational reinforcement learning: an overview”. In: *Proceedings of the ICML-2004 Workshop on Relational Reinforcement Learning*, pp. 1–9.

Tang, J., J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su (2008). “ArnetMiner: extraction and mining of academic social networks”. In: *KDD’08*, pp. 990–998 (p. 60).

Taskar, B., P. Abbeel, and D. Koller (2002). “Discriminative probabilistic models for relational data”. In: *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI), (2002)*. UAI’02. Alberta, Canada, pp. 485–492.

Teso, S., R. Sebastiani, and A. Passerini (2013). “Hybrid SRL with optimization modulo theories”. In: *2013 NIPS Workshop on Constructive Machine Learning, Lake Tahoe, Nevada, USA* (pp. 83, 108).

Ullmann, J. R. (1976). “An algorithm for subgraph isomorphism”. In: *Journal of the ACM* 23(1), pp. 31–42 (pp. 49, 71).

Valiant, L. G. (1979). “The complexity of computing the permanent”. In: *Theoretical computer science* 8(2), pp. 189–201 (p. 40).

Van Assche, A., C. Vens, H. Blockeel, and S. Džeroski (2006). “First order random forests: learning relational classifiers with complex aggregates”. In: *Machine Learning* 64(1-3), pp. 149–182 (pp. 37, 132).

Van den Broeck, G., W. Meert, and J. Davis (2013). “Lifted generative parameter learning”. In: *Statistical Relational Artificial Intelligence, Papers from the 2013 AAAI Workshop, Bellevue, Washington, USA, July 15, 2013* (p. 136).

Van Emden, M. H. and R. A. Kowalski (1976). “The semantics of predicate logic as a programming language”. In: *Journal of the ACM (JACM)* 23(4), pp. 733–742.

Van Haaren, J., G. Van den Broeck, W. Meert, and J. Davis (2015). “Lifted generative learning of Markov logic networks”. In: *Machine Learning*, pp. 1–29 (p. 136).

Vens, C., J. Ramon, and H. Blockeel (2006). “ReMauve: a relational model tree learner”. In: *International Conference on Inductive Logic Programming*. Springer, pp. 424–438 (p. 37).

Venugopal, D., S. Sarkhel, and V. Gogate (2015). "Just count the satisfied groundings: scalable local-search and sampling based inference in MLNs." In: *AAAI*. Citeseer, pp. 3606–3612 (p. 72).

Vianna, L. G. R., L. N. de Barros, and S. Sanner (2015). "Real-time symbolic dynamic programming for hybrid MDPs". In: *Proceedings of the 29th Conference on Artificial Intelligence (AAAI), (2015)*. AAAI'15. Austin, Texas, pp. 3402–3408.

Vien, N. A. and M. Toussaint (2013). "Reasoning with uncertainties over existence of objects". In: *AAAI Fall Symposium: How Should Intelligence Be Abstracted in AI Research* (p. 138).

Wang, C. and R. Khardon (2010). "Relational partially observable MDPs". In: *Proceedings of the 24th Conference on Artificial Intelligence (AAAI), (2010)*. Vol. 10, pp. 1153–1157.

Wang, J. and P. Domingos (2008). "Hybrid Markov logic networks". In: *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI), (2008)*. AAAI'08. Chicago, Illinois, pp. 1106–1111. (Pp. 83, 108).

Wernicke, S. (2005). "A faster algorithm for detecting network motifs". In: *Proceedings of the 5th International conference on Algorithms in Bioinformatics*. WABI'05. Mallorca, Spain, pp. 165–177 (p. 71).

Wiering, M. and M. Van Otterlo (2012). "Reinforcement learning". In: *Adaptation, Learning, and Optimization* 12.

Yan, X. and J. Han (2002). "Span: graph-based substructure pattern mining". In: *Proceedings of the 2002 IEEE International Conference on Data Mining*. ICDM '02. Washington, DC, USA, pp. 721–724 (p. 71).

Yang, S., T. Khot, K. Kersting, and S. Natarajan (2016). *Learning continuous-time Bayesian networks in relational domains: A non-parametric approach*.

Yuan, C. and M. J. Druzdzel (2007). "Importance sampling for general hybrid Bayesian networks". In: *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AISTATS-07)*. Vol. 2, pp. 652–659 (p. 107).

Zou, R. and L. B. Holder (2010). "Frequent subgraph mining on a single large graph using sampling techniques". In: *Proceedings of the 8th Workshop on Mining and Learning with Graphs (2010)*. MLG '10. New York, NY, USA, pp. 171–178 (pp. 71, 72).

Curriculum Vitae

Irma Ravkic was born in Tuzla, Bosnia and Herzegovina on February 2nd 1986. She attended "Mesa Selimovic" linguistic gymnasium in Tuzla and continued her higher education in 2004 at the University of Tuzla. In 2009 she obtained a Bachelor of Electrical engineering degree with a Golden plaquette distinction. In 2011 she receives a JoinEU-See scholarship funded by the European Commission for student mobility between Western Europe and the Balkans. This scholarship enabled her to graduate as a Master of Science specialized in Artificial Intelligence at KU Leuven in Belgium with magna cum laude distinction. Her master thesis titled "Iris segmentation using monogenic filter" was about a specific application of computer vision to iris recognition.

Supported by the Research Fund KU Leuven (OT/11/051), in 2011 she began her doctoral studies in the Machine Learning group at the DTAI lab (Declarative Languages and Artificial Intelligence) under the supervision of prof. dr. Jesse Davis and co-supervision of dr. ir. Jan Ramon. During her doctoral studies she was active in the local student community as the ombuds and a representative of several Master of Artificial Intelligence programme committees. She was also active in promoting computer science through various creative channels such as winning the Best Video Prize at the Intelligent Data Analysis conference in 2014, and translating code.org tutorials from English to her native language. These and similar activities were important factors in her being one of the recipients of the Google Anita Borg 2015 memorial scholarship for women excelling in computer science. In October 2016 she will defend her doctoral thesis, titled "Probabilistic Logical Models for Large-Scale Hybrid Domains".

List of Publications

Journal Articles

Ravkic, I., J. Ramon, and J. Davis (2015). “Learning relational dependency networks in hybrid domains”. In: *Machine Learning* 100(2), pp. 217–254. (Pp. 6, 114, 131).

Ravkic, I., M. Znidarsic, J. Ramon, and J. Davis (2016). “Graph sampling for efficient parameter estimation in statistical relational learning”. In: *(under revision) ECML-PKDD Data mining and Knowledge discovery journal*.

Conference Articles

Nitti, D., I. Ravkic, J. Davis, and L. De Raedt (2016). “Learning the structure of dynamic hybrid relational models”. In: *22nd European Conference on Artificial Intelligence (ECAI) 2016, European Conference on Artificial Intelligence (ECAI), The Hague, The Netherlands, 29 - 2 September 2016*, pp. 1283–1290.

Ravkic, I., J. Ramon, and J. Davis (2012). “Hybrid logical Bayesian networks”. In: *Late Breaking Papers proceedings of the 22nd International Conference on Inductive Logic Programming 62-67* (p. 4).

Workshop Papers

Ravkic, I., B. Wittevrongel, W. Meert, J. Davis, T. A. Gerbrands, and B. Vanwanseele (2015). “Predicting gait retraining strategies for knee osteoarthritis”. In: Workshop on Machine Learning in Life Sciences, ECML-PKDD 2015, Porto, Portugal (p. 12).

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
DTAI
Celestijnenlaan 200A box 2402
B-3001 Leuven
irma.ravkic@cs.kuleuven.be