Structure Learning in Markov Logic Networks

Stanley Kok

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

2010

Program Authorized to Offer Degree: Computer Science & Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Stanley Kok

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of the Supervisory Committee:

_____

Pedro Domingos

Reading Committee:

_____

Pedro Domingos

_____

Oren Etzioni

_____

Daniel S. Weld

Date: _____

In presenting this dissertation in partial fulfillment of the requirements for the doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Proquest Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, 1-800-521-0600, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature_____

Date_____

University of Washington

Abstract

Structure Learning in Markov Logic Networks

Stanley Kok

Chair of the Supervisory Committee:
Professor Pedro Domingos
Computer Science & Engineering

Markov logic networks (MLNs) [86, 24] are a powerful representation combining first-order logic and probability. An MLN attaches weights to first-order formulas and views these as templates for features of Markov networks. Learning MLN structure consists of learning both formulas and their weights. This is a challenging problem because of its super-exponential search space of formulas, and the need to repeatedly learn the weights of formulas in order to evaluate them, a process that requires computationally expensive statistical inference. This thesis presents a series of algorithms that efficiently and accurately learn MLN structure.

We begin by combining ideas from inductive logic programming (ILP) and feature induction in Markov networks in our MSL system. Previous approaches learn MLN structure in a disjoint manner by first learning formulas using off-the-shelf ILP systems and then learning formula weights that optimize some measure of the data's likelihood. We present an integrated approach that learns both formulas and weights that jointly optimize likelihood.

Next we present the MRC system that learns *latent* MLN structure by discovering unary predicates in the form of clusters. MRC forms multiple clusterings of constants and relations, with each cluster corresponding to an invented predicate. We empirically show that by creating multiple clusterings, MRC outperforms previous systems.

Then we apply a variant of MRC to the long-standing AI problem of extracting knowledge from text. Our system extracts simple semantic networks in an unsupervised, domain-independent manner from Web text, and introduces several techniques to scale up to the Web.

After that, we incorporate the discovery of latent unary predicates into the learning of MLN clauses in the LHL system. LHL first compresses the data into a compact form by clustering the constants into high-level concepts, and then searches for clauses in the compact representation. We empirically show that LHL is more efficient and finds better formulas than previous systems.

Finally, we present the LSM system that makes use of *random walks* to find repeated patterns in data. By restricting its search to within such patterns, LSM is able to accurately and efficiently find good formulas, improving efficiency by 2-5 orders of magnitude compared to previous systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Pedro Domingos, for your guidance and mentorship. Without you, this thesis would not have been possible. You taught me what it takes to be a good researcher.

Thanks also go to Oren Etzioni. I appreciate your advice and encouragement during my early research work as a graduate student. Your interest in creating useful and intelligent AI applications has definitely rubbed off on me.

Finally, I would like to thank Dan Weld for the advice you provided as my first-year temporary advisor. It is because of your encouragement that I signed up for numerous AI-related courses, thereby expanding my view of the field.

DEDICATION

To my parents and maternal grandparents

Chapter 1

INTRODUCTION

Statistical learning handles uncertainty in a robust and principled way. Relational learning (also known as inductive logic programming (ILP)) models domains involving multiple relations. Recent years have seen a surge of interest in the statistical relational learning (SRL) community in combining the two, driven by the realization that many (if not most) applications require both [36].

Markov logic networks (MLNs) [86, 24] are a type of statistical relational model that has gained traction within the AI community in recent years because of its robustness to noise and its ability to model complex domains. MLNs combine probability and logic by attaching weights to first-order formulas [35], and viewing these as templates for features of Markov networks [75]. Learning the structure of an MLN consists of learning both formulas and their weights.

Learning MLN structure from data is an important task because it allows us to discover novel knowledge. The need for it becomes especially acute when the data is too large for human perusal, and we lack expert knowledge about it. For example, from a large database describing a university, we would want a system to automatically discover probabilistic rules capturing the relationships among professors, students, courses, etc. These rules could then be used for tasks such as predicting future student enrollment in courses.

MLN structure learning is an extremely challenging task because of its super-exponential search space of formulas, and its need to repeatedly learn the weights of formulas in order to evaluate them, a process that requires computationally expensive statistical inference.

This thesis addresses the question of how to efficiently and accurately learn MLN strucure from relational data (Figure 1.1). We begin by describing the Markov logic Structure Learner (MSL) [45] that combines ideas from ILP and feature induction in Markov networks. We show that by combining the strengths of both relational and statistical approaches, MSL outperforms systems that use only one of the two approaches.

Next we propose statistical predicate invention (SPI) as a key problem for statistical relational

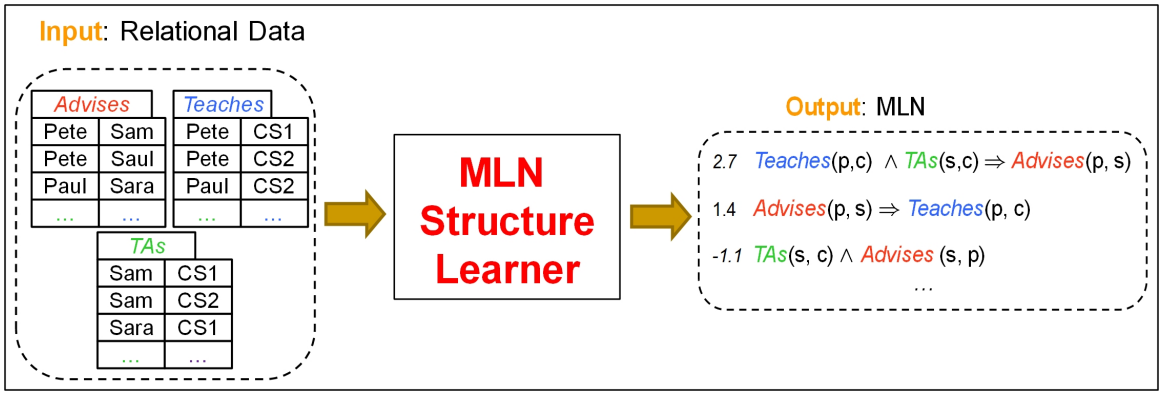

Figure 1.1: Input and output of MLN structure learner.

learning. SPI is the problem of discovering new concepts, properties and relations in structured data, and generalizes hidden variable discovery in statistical models and predicate invention in ILP. We present an approach for learning *latent* MLN structure as an initial model for SPI. Our algorithm, Multiple Relational Clusterings (MRC) [46], is based on second-order Markov logic, in which predicates as well as arguments can be variables, and the domain of discourse is not fully known in advance. Our approach iteratively refines clusters of symbols based on the clusters of symbols they appear in atoms with (e.g., it clusters relations by the clusters of the objects they relate). Since different clusterings are better for predicting different subsets of the atoms, we allow multiple cross-cutting clusterings. We demonstrate that by discovering new concepts and relations, MRC outperforms MSL and other comparison systems on a number of relational datasets.

After that, we apply the learning of latent MLN structure to the long-standing AI problem of extracting knowledge from text. We create the Semantic Network Extractor (SNE) system [47] that learns semantic networks from Web data in an unsupervised, domain-independent manner. SNE simultaneously clusters phrases into high-level concepts and relations, and discovers interactions among them in the form of a semantic network. To scale to the Web, SNE incorporates several techniques to find the clusters efficiently. Our empirical evaluation shows that SNE is able to extract meaningful semantic networks from a large Web corpus.

Next we incorporate the discovery of latent structure into the learning of MLN formulas in the Learning via Hypergraph Lifting (LHL) system [48]. LHL views a relational database as a hyper-

graph with constants as nodes and relations as hyperedges. It finds paths of true ground atoms in the hypergraph that are connected via their arguments. To make this tractable (there are exponentially many paths in the hypergraph), LHL *lifts* the hypergraph by jointly clustering the constants to form higher-level concepts and finds paths in it. The paths are then converted into weighted formulas. We empirically show that LHL learns more accurate rules than previous systems.

Finally, we address the problem of learning long MLN formulas. Learning long formulas is important for two reasons. First, long formulas can capture more complex dependencies in data than short ones. Second, when we lack domain knowledge, we typically want to set the maximum formula length to a large value so as not to *a priori* preclude any good rule. State-of-the-art MLN structure learners are only able to learn short formulas (4-5 literals) due to the extreme computational cost of learning. We create the Learning via Structural Motifs (LSM) system [49] that efficiently and accurately learns long formulas by constraining its search to within frequently occurring patterns called *structural motifs*. Our experiments demonstrate that LSM is 2-5 orders of magnitude faster than previous systems, while achieving the same or better predictive performance.

1.1 Thesis Overview

The next chapter reviews the necessary background on first-order logic, Markov networks and Markov logic. Chapter 3 describes how ideas from ILP and feature induction in Markov networks are combined in the MSL system. Chapter 4 and Chapter 5 respectively present latent structure discovery in the MRC and SNE systems. Chapter 6 shows how the LHL system 'lifts' hypergraphs to find good rules. Chapter 7 describes the motif discovery algorithm of the LSM system, and how motifs are used to find long formulas. Each chapter discusses related work. Finally, the thesis concludes with a summary of its contributions and directions for future work.

Chapter 2

BACKGROUND

In this chapter, we provide background on first-order logic and Markov networks, and then describe how Markov logic unifies the two concepts.

2.1 First-Order Logic

In first-order logic [35], formulas are constructed using the following four types of symbols.

- **Constants** represent objects in a domain of discourse (e.g., people: `Anna`, `Bob`, `Charles`).
- **Variables** (e.g., `x`, `y`, `z`) range over the objects in the domain.
- **Functions** (e.g., `FatherOf`, `GreatestCommonDivisorOf`) represent mappings from tuples of objects to objects.
- **Predicates** represent relations among objects (e.g., `Friends`, `Advises`), or attributes of objects (e.g., `Tall`, `IsSmoker`).

An *interpretation* specifies which objects, functions and relations in the domain are represented by which symbols. Variables and constants may be *typed*, in which case variables range only over objects of the corresponding type, and constants can only represent objects of the corresponding type.

A *term* is an expression representing an object in the domain, and can be a constant, a variable, or a function applied to a tuple of terms (e.g., `Anna`, `x` and `FatherOf(x)`). An *atom* is a predicate symbol applied to a tuple of terms (e.g., `Friends(x, Anna)`, `FatherOf(Anna)`). A *positive literal* is an atom; a *negative literal* is a negated atom. A *ground term* is a term containing no variables. A *ground atom* or *ground predicate* is an atom all of whose arguments are ground terms.

Formulas F and F' are recursively constructed from atoms using logical connectives and quantifiers in the following manner.

- **Conjunction**. $F \wedge F'$, which is true iff both F and F' are true.
- **Disjunction**. $F \vee F'$, which is true iff F or F' is true.
- **Negation**. $\neg F$, which is true iff F is false.
- **Implication**. $F \Rightarrow F'$, which is true iff F is false or F' is true.
- **Equivalence**. $F \Leftrightarrow F'$, which is true iff F and F' have the same truth value.
- **Existential Quantification**. $\exists \mathtt{x}\ F$, which is true iff F is true for at least one object $\mathtt{x}$ in the domain.
- **Universal Quantification**. $\forall \mathtt{x}\ F$, which is true iff F is true for every object $\mathtt{x}$ in the domain.

A *clause* is a disjunction of positive/negative literals. A *definite clause* is a clause with exactly one positive literal (the *head*, with negative literals constituting the *body*). Every first-order formula can be converted into an equivalent formula in *conjunctive normal form*, $Qx_1 \ldots Qx_n\ C(x_1, \ldots, x_n)$, where each Q is a quantifier, each x_i is a quantified variable, and $C(\ldots)$ is a conjunction of clauses.

A *first-order knowledge base (KB)* is a set of formulas in first-order logic. The formulas in a KB are implicitly conjoined, and thus a KB can be viewed as a single large formula. A KB in *clausal form* is a conjunction of clauses.

A *world* is an assignment of truth values to all possible ground atoms, and thus to every formula in the KB. A *database* is a partial specification of a world; each atom in it is true, false or (implicitly) unknown. In this thesis, unless stated otherwise, we make the *closed-world* assumption, i.e., all ground atoms not in the database are assumed false.

The main inference problem in first-order logic is to determine whether a KB *entails* a formula F, i.e., if F is true in all worlds where the KB is true. This problem is semidecidable.

First-order logic has limited practical applicability to modeling real-world domains for two reasons. First, if a KB contains a contradiction (common in real-world domains where a formula and its negation can be true under different circumstances), all formulas are trivially entailed by it. Thus, a first-order KB requires painstaking knowledge engineering. This problem becomes even more pronounced when different KBs are merged to capture a wider range of knowledge. Second, in most domains it is difficult to specify non-trivial formulas that are always true, and such formulas only

represent a small portion of the relevant knowledge. We shall show in Section 2.3 how Markov logic overcomes these limitations.

Inductive logic programming (ILP) systems learn clausal KBs from relational databases, or refine existing KBs [54]. In the *learning from entailment* setting, the system searches for clauses that entail all positive examples of some relation (e.g., `Friends`) and no negative ones. For example, FOIL [80] learns each clause by starting with the target relation as the head and greedily adding literals to the body, using an information-theoretic measure to choose among candidate literals. In the *learning from interpretations* setting, the examples are databases, and the system searches for clauses that are true in them. For example, CLAUDIEN [18], starting with a trivially false clause (`true` $\Rightarrow$ `false`), repeatedly forms all possible refinements of the current clauses by adding literals to the head and body, and adds to the KB the ones that satisfy a minimum accuracy and coverage criterion.

2.2 Markov Networks

A *Markov network* or *Markov random field* [75] is a model for the joint distribution of a set of variables $X = (X_1, X_2, \ldots, X_n) \in \mathcal{X}$. It is composed of an undirected graph G and a set of potential functions ϕ_k. The graph has a node for each variable, and the model has a potential function for each clique in the graph. A potential function is a non-negative real-valued function of the state of the corresponding clique. The joint distribution represented by a Markov network is given by

$$P(X=x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}}) \tag{2.1}$$

where $x_{\{k\}}$ is the state of the kth clique (i.e., the state of the variables that appear in that clique). Z, known as the *partition function*, is given by $Z = \sum_{x \in \mathcal{X}} \prod_k \phi_k(x_{\{k\}})$. Markov networks are often conveniently represented as *log-linear models*, with each clique potential replaced by an exponentiated weighted sum of features of the state, leading to

$$P(X=x) = \frac{1}{Z} \exp\left(\sum_j w_j f_j(x) \right). \tag{2.2}$$

A feature may be any real-valued function of the state. This thesis will focus on binary features ($f_j(x) \in \{0, 1\}$). In the most direct translation from the potential-function form, there is one feature corresponding to each possible state $x_{\{k\}}$ of each clique, with its weight being $\log \phi_k(x_{\{k\}})$. This representation is exponential in the size of the cliques. However, we are free to specify a much smaller number of features (e.g., logical functions of the state of the clique), allowing for a more compact representation than the potential-function form, particularly when large cliques are present. As we shall show in the next section, Markov logic takes advantage of this.

Approximate inference is widely used in place of exact inference in Markov networks because the latter is #P-complete [88]. The most commonly used approximate method is Markov chain Monte Carlo (MCMC) [37], in particular Gibbs sampling. Gibbs sampling works by sampling each variable x in turn given its Markov blanket, which is defined as the minimal set of variables that makes x independent of all other variables. (In a Markov network, x's Markov blanket simply contains its neighbors in the graph.) Marginal probabilities are computed by counting over these samples, and conditional probabilities are computed by sampling with the conditioning variables fixed to their given values. A drawback of MCMC is that it can be very slow to converge. Another widely used method for inference is belief propagation [108]. It operates by first constructing a bipartite graph of the nodes and the potentials. Then it passes approximations to node marginals as messages from variable nodes to their corresponding factor nodes and vice versa. A disadvantage of such a message-passing scheme is that it does not provide any guarantee of convergence or of giving correct marginals when it converges.

Maximum-likelihood estimates of Markov network weights cannot be computed in closed form, but they can be found using standard gradient-based or quasi-Newton optimization methods like L-BFGS [73] (because the log-likelihood is a concave function of the weights). Since such methods require inference as subroutines, they inherit their subroutines' computational costs and drawbacks.

The standard approach to learning the structure (i.e., the features) of Markov networks is introduced by Della Pietra et al. (1997). They induce conjunctive features by starting with a set of atomic features (the original variables), conjoining each current feature with each atomic feature, adding to the network the conjunction that most increases likelihood, and repeating. McCallum (2003) extends this to the case of conditional random fields, which are Markov networks trained to maximize the conditional likelihood of a set of outputs given a set of inputs.

More recently, Lee et al. (2007) and Ravikumar et al. (2009) learn Markov network structure via weight learning with L_1 priors. An L_1 prior has the property of forcing weights to zero by penalizing small weights severely. These approaches consider the space of all possible features and find their L_1-regularized weights that optimize the log-likelihood of data. Features with non-zero weights are retained in the Markov network, and the rest are discarded. Even though, conceptually, these models can work over all possible features, they have only been used in practice for features containing at most two variables for tractability reasons.

2.3 Markov Logic

A first-order KB can be seen as a set of hard constraints on the set of possible worlds: if a world violates even one formula, it has zero probability. The basic idea in Markov logic [86, 24] is to soften these constraints: when a world violates one formula in the KB it is less probable, but not impossible. The fewer formulas a world violates, the more probable it is. Each formula has an associated weight that reflects how strong a constraint it is: the higher the weight, the greater the difference in log probability between a world that satisfies the formula and one that does not, other things being equal.

Definition 1. [86, 24] *A Markov logic network L is a set of pairs (F_i, w_i), where F_i is a formula in first-order logic and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2, \ldots, c_{|C|}\}$, it defines a Markov network $M_{L,C}$ (Equations 2.1 and 2.2) as follows:*

1. *$M_{L,C}$ contains one binary node for each possible grounding of each predicate appearing in L. The value of the node is 1 if the ground predicate is true, and 0 otherwise.*

2. *$M_{L,C}$ contains one feature for each possible grounding of each formula F_i in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the w_i associated with F_i in L.*

Thus there is an edge between two nodes of $M_{L,C}$ iff the corresponding ground predicates appear together in at least one grounding of one formula in L.

A Markov logic network (MLN) can be viewed as a *template* for constructing Markov networks. For different sets of constants, an MLN can construct Markov networks of varying sizes, all of which share regularities in structure and parameters as given by an MLN. From Definition 1 and Equation 2.2, the probability distribution over possible worlds x specified by the Markov network $M_{L,C}$ is given by

$$P(X=x) = \frac{1}{Z} \exp\left(\sum_{i \in F} \sum_{j \in G_i} w_i g_j(x)\right) \tag{2.3}$$

where Z is a normalization constant, F is the set of all first-order formulas in the MLN L, G_i and w_i are respectively the set of groundings and weight of the ith first-order formula, and $g_j(x) = 1$ if the jth ground formula is true and $g_j(x) = 0$ otherwise. As formula weights increase, an MLN increasingly resembles a purely logical KB, becoming equivalent to one in the limit of all infinite weights. Markov logic allows contradictions between formulas simply by weighting the formulas and weighing evidence on both sides. Markov logic can also represent complex non-i.i.d. models (which do not assume that data points are independent and identically distributed) by allowing a predicate to appear more than once in a formula. This permits information to propagate among the different occurrences of the predicate.

Markov logic has as special cases all discrete probabilistic models that are expressible as products of potentials (including Markov networks and Bayesian networks). For details on how Markov logic is related to other statistical relational models, we refer the reader to Richardson and Domingos (2006).

In this thesis, we focus on MLNs whose formulas are function-free clauses, and assume domain closure (i.e., the only objects in the domain are those representable using the constant symbols in C), thereby ensuring that the Markov networks generated are finite.

MLN weights can in principle be learned using standard gradient-based or quasi-Newton optimization methods [73] because the log-likelihood (i.e., log of Equation 2.3) is a concave function of the weights. The derivative of the log-likelihood with respect to the weight of the ith formula is

$$\frac{\partial}{\partial w_i} \log P_w(X=x) = n_i(x) - \sum_{x'} P_w(X=x')\, n_i(x') \tag{2.4}$$

where the sum is over all possible databases x', $n_i(x)$ is the number of true groundings of the ith formula in the data x, and $P_w(X = x')$ is $P(X = x')$ computed using the weight vector $w = (w_1, \ldots, w_i, \ldots)$. In other words, the ith component of the gradient is simply the difference between the number of true groundings of the ith formula in the data and its expectation according to the current model. Unfortunately, computing the expectation requires summing over all possible databases, which is intractable. Furthermore, quasi-Newton optimization methods require computing the log-likelihood and thus its partition function Z, which is also intractable. Even though Markov chain Monte Carlo techniques can be used to approximate the expectation and partition function, they are computationally expensive.

A more efficient, widely used alternative is to optimize pseudo-log-likelihood [5].

$$\log P^*_{w,F}(X{=}x) = \sum_{g=1}^{n} \log P_{w,F}(X_g{=}x_g | MB_x(X_g)) \tag{2.5}$$

where F is the set of all first-order formulas in an MLN, $MB_x(X_g)$ is the state of the Markov blanket of X_g in the data (i.e., the truth values of the ground atoms it appears in some ground formula with), and

$$P_{w,F}(X_g{=}x_g | MB_x(X_g)) = \frac{\exp\left(\sum_{i=1}^{F} w_i n_i(x)\right)}{\exp\left(\sum_{i=1}^{F} w_i n_i(x_{[X_g=0]}\right) + \exp\left(\sum_{i=1}^{F} w_i n_i(x_{[X_g=1]}\right)} \tag{2.6}$$

where $n_i(x)$ is the number of true groundings of the ith formula in x, $n_i(x_{[X_g=0]})$ is the number of true groundings of the ith formula when we force $X_g = 0$ and leave the remaining data unchanged, and similarly for $n_i(x_{[X_g=1]})$.

Pseudo-log-likelihood and its gradient (given below) do not require inference over the model.

$$\begin{aligned} \frac{\partial}{\partial w_i} \log P^*_{w,F}(X{=}x) \;\; = \;\; & \sum_{g=1}^{n} [\; n_i(x) - P_{w,F}(X_g{=}0 | MB_x(X_g))\; n_i(x_{[X_g=0]}) \\ & - P_{w,F}(X_g{=}1 | MB_x(X_g))\; n_i(x_{[X_g=1]}) \;]. \end{aligned} \tag{2.7}$$

Richardson and Domingos (2006) used an off-the-shelf ILP system (CLAUDIEN [18]) to learn MLN clauses. However, since an MLN represents a probability distribution, a sounder approach

would be to use a likelihood-based evaluation function, rather than typical ILP ones like accuracy and coverage, to guide the creation of MLN clauses. In Chapter 3, we present a such an approach.

Chapter 3

MARKOV LOGIC STRUCTURE LEARNER

3.1 Introduction

In this chapter, we develop an algorithm for learning the structure of MLNs from relational databases, combining ideas from inductive logic programming (ILP; Section 2.1) and feature induction in Markov networks (Section 2.2).

The previous MLN structure learning approach, proposed by Richardson and Domingos (2006), used an off-the-shelf ILP system (CLAUDIEN [18]) to induce first-order clauses, and then learned maximum pseudo-likelihood weights for them. This is unlikely to give the best results, because CLAUDIEN (like other ILP systems) is designed to simply find clauses that hold with some accuracy and frequency in the data, not to maximize the data's likelihood (and hence the quality of the MLN's probabilistic predictions).

We develop the Markov logic Structure Learner (MSL) system for learning the structure of MLNs by directly optimizing a likelihood-type measure, and show experimentally that it outperforms the approach of Richardson and Domingos. The algorithm performs a beam or shortest-first search of the space of clauses, guided by a weighted pseudo-likelihood measure. This requires computing the optimal weights for each candidate structure, but we show how this can be done efficiently. The algorithm can be used to learn an MLN from scratch, or to refine an existing knowledge base. Either way, like Richardson and Domingos (2006), we have found it useful to start by adding all unit clauses (single predicates) to the MLN. The weights of these capture (roughly speaking) the marginal distributions of the predicates, allowing the longer clauses to focus on modeling predicate dependencies.

The design space for MLN structure learning algorithms includes the choice of evaluation measure, clause construction operators, search strategy and speedup methods. We discuss each of these in turn in the next four sections. In Section 3.6, we report our experiments on two real-world domains, which show that MSL outperforms using off-the-shelf ILP systems to learn MLN structure,

as well as purely ILP, purely probabilistic and purely knowledge-based approaches. We discuss related work in Section 3.7.

3.2 Evaluation Measures

We initially used the same pseudo-likelihood measure as Richardson and Domingos (Equation 2.5). However, we found this to give undue weight to the predicate with the largest number of groundings (typically the largest-arity predicate), resulting in poor modeling of the rest. We thus defined the weighted pseudo-log-likelihood (WPLL) as

$$\log P^{\bullet}_{w,F,D}(X=x) = \sum_{r \in R} c_r \sum_{g \in G^D_r} \log P_{w,F}(X_g = x_g | MB_x(X_g)) \tag{3.1}$$

where F is a set of clauses, w is a set of clause weights, R is the set of first-order predicates, G^D_r is a set of ground atoms of predicate r in database D, and x_g is the truth value (0 or 1) of ground atom g in D, and $P_{w,F}(X_g = x_g | MB_x(X_g))$ is given by Equation 2.6. The choice of predicate weights c_r depends on the user's goals. In our experiments, we simply set $c_r = 1/|G^D_r|$ which has the effect of weighting all first-order predicates equally. If modeling a predicate is not important (e.g., because it will always be part of the evidence), we set its weight to zero. We used WPLL in all versions of MLN learning in our experiments. To combat overfitting, we penalize the WPLL with a structure prior of $e^{-\pi \sum_{i=1}^{F} d_i}$, where d_i is the number of predicates that differ between the current version of the clause and the original one (if the clause is new, this is simply its length), and π is the penalty per predicate. This is similar to the approach used in learning Bayesian networks [40]. Following Richardson and Domingos, we also penalize each weight with a Gaussian prior.

A potentially serious problem that arises when evaluating candidate clauses using WPLL is that the optimal (maximum WPLL) weights need to be computed for each candidate. Given that this involves numerical optimization, and may need to be done thousands or millions of times, it could easily make the algorithm too slow to be practical. Indeed, in the UW-CSE domain (see Section 3.6), we found that learning the weights using L-BFGS took 3 minutes on average, which is fast enough if only done once, but infeasible to do for every candidate clause.

Della Pietra et al. (1997) and McCallum (2003) address this problem by assuming that the weights of previous features do not change when testing a new one. Surprisingly, we found this to

be unnecessary if we use the very simple approach of initializing L-BFGS with the current weights (and zero weight for a new clause). Although in principle all weights could change as the result of introducing or modifying a clause, in practice this seldom happens. Second-order, quadratic-convergence methods like L-BFGS are known to be very fast if started near the optimum. This is what happens in our case; L-BFGS typically converges in just a few iterations, sometimes one. The time required to evaluate a clause is in fact dominated by the time required to compute the number of its true groundings in the data, and this is a problem we focus on in Section 3.5.

3.3 Clause Construction Operators

When learning an MLN from scratch (i.e., from a set of unit clauses), the natural operator to use is the addition of a literal to a clause. When refining a hand-coded KB, the goal is to correct the errors made by the human experts. These errors include omitting conditions from rules and including spurious ones, and can be corrected by operators that add and remove literals from a clause. These are the basic operators that we use. In addition, we have found that many common errors (e.g., wrong direction of implication) can be corrected at the clause level by flipping the signs of predicates, and we also allow this. When adding a literal to a clause, we consider all possible ways in which the literal's variables can be shared with existing ones, subject to the constraint that the new literal must contain at least one variable that appears in an existing one. To control the size of the search space, we set a limit on the number of distinct variables in a clause. We only try removing literals from the original hand-coded clauses or their descendants, and we only consider removing a literal if it leaves at least one path of shared variables between each pair of remaining literals.

3.4 Search Strategies

We have implemented two search strategies, one faster and one more complete. The first approach adds clauses to the MLN one at a time, using beam search to find the best clause to add: starting with the unit clauses and the expert-supplied ones, we apply each legal literal addition and deletion to each clause, keep the b best ones, apply the operators to those, and repeat until no new clause improves the WPLL. The chosen clause is the one with highest WPLL found in any iteration of the search. If the new clause is a refinement of a hand-coded one, it replaces it. (Notice that, even though

we both add and delete literals, no loops can occur because each change must improve WPLL to be accepted.)

The second approach adds k clauses at a time to the MLN, and is similar to that of McCallum (2003). In contrast to beam search, which adds the best clause of any length found, this approach adds all "good" clauses of length l before attempting any of length $l + 1$. We call it *shortest-first search*.

Table 3.1 shows the structure learning algorithm in pseudo-code, Table 3.2 shows beam search, and Table 3.3 shows shortest-first search for the case where the initial MLN contains only unit clauses.

3.5 Speedup Techniques

The algorithms described in the previous section may be very slow, particularly in large domains. However, they can be greatly sped up using a combination of techniques that we now describe.

- Richardson and Domingos (2006) list several ways of speeding up the computation of the pseudo-log-likelihood and its gradient, and we apply them to the WPLL (Equation 3.1). In addition, in Equation 2.6 we ignore all clauses that the predicate does not appear in.

- When learning MLN weights to evaluate candidate clauses, we use a looser convergence threshold and lower maximum number of iterations for L-BFGS than when updating the MLN with the chosen clause(s).

- We compute the contribution of a predicate to the WPLL approximately by uniformly sampling a fraction of its groundings (true and false ones separately), computing the conditional likelihood of each one (Equation 2.6), and extrapolating the average. The number of samples can be chosen to guarantee that, with high confidence, the chosen clause(s) are the same that would be obtained if we computed the WPLL exactly. At the end of the algorithm we do a final round of weight learning without subsampling.

- We use a similar strategy to compute the number of true groundings of a clause, required for the WPLL and its gradient. In particular, we use the algorithm of Karp and Luby (1983). In

Table 3.1: MSL algorithm.

---

function *StructLearn*(R, MLN, DB)

inputs: R, a set of predicates

MLN, a clausal Markov logic network

DB, a relational database

output: Modified MLN

Add all unit clauses from R to MLN

for each non-unit clause c in MLN (optionally)

Try all combinations of sign flips of literals in c, and keep the one that gives the highest *WPLL*(MLN, DB)

$Clauses_0 \leftarrow \{$All clauses in $MLN\}$

LearnWeights(MLN, DB)

$Score \leftarrow$ *WPLL*(MLN, DB)

repeat

$Clauses \leftarrow$ *FindBestClauses*(R, MLN, $Score$, $Clauses_0$, DB)

if $Clauses \neq \emptyset$

Add $Clauses$ to MLN

LearnWeights(MLN, DB)

$Score \leftarrow$ *WPLL*(MLN, DB)

until $Clauses = \emptyset$

for each non-unit clause c in MLN

Prune c from MLN unless this decreases *WPLL*(MLN, DB)

return MLN

Table 3.2: Beam search for the best clause.

---

function *FindBestClauses*(R, MLN, $Score$, $Clauses_0$, DB)

inputs: R, a set of predicates

MLN, a clausal Markov logic network

$Score$, WPLL of MLN

$Clauses_0$, a set of clauses

DB, a relational database

output: $BestClause$, a clause to be added to MLN

$BestClause \leftarrow \emptyset$

$BestGain \leftarrow 0$

$Beam \leftarrow Clauses_0$

Save the weights of the clauses in MLN

repeat

$Candidates \leftarrow$ *CreateCandidateClauses*($Beam$, R)

for each clause $c \in Candidates$

Add c to MLN

LearnWeights(MLN, DB)

$Gain(c) \leftarrow$ *WPLL*(MLN, DB) $- Score$

Remove c from MLN

Restore the weights of the clauses in MLN

$Beam \leftarrow \{$The b clauses $c \in Candidates$ with highest $Gain(c) > 0$ & $Weight(c) > \epsilon > 0$ $\}$

if $Gain$(Best clause c^* in $Beam$) $> BestGain$

$BestClause \leftarrow c^*$

$BestGain \leftarrow Gain(c^*)$

until $Beam = \emptyset$ or $BestGain$ has not changed in two iterations

return $\{BestClause\}$

Table 3.3: Shortest-first search for the best clauses.

function *FindBestClauses*($R, MLN, Score, Clauses_0, DB$)

inputs: R, a set of predicates

MLN, a clausal Markov logic network

$Score$, WPLL of MLN

$Clauses_0$, a set of clauses

DB, a relational database

output: $BestClauses$, a set of clauses to be added to MLN

Save the weights of the clauses in MLN

if this is the first time *FindBestClauses* is called

$Candidates \leftarrow \emptyset$

$l \leftarrow 1$

repeat

if $l = 1$ or this is not the first iteration of the repeat loop

if there is no clause in $Candidates$ of length $< l$ that was not previously extended

$l \leftarrow l + 1$

$Clauses \leftarrow$ {Clauses of length $l-1$ in MLN not previously extended } $\cup$

{s best clauses of length $l-1$ in $Candidates$ not previously extended }

$Candidates \leftarrow Candidates \cup$ *CreateCandidateClauses*($Clauses$, R)

for each clause $c \in Candidates$ not previously evaluated

Add c to MLN

LearnWeights(MLN, DB)

$Gain(c) \leftarrow$ *WPLL*(MLN, DB) $- Score$

Remove c from MLN and restore the weights of the clauses in MLN

$Candidates \leftarrow \{m$ best clauses in $Candidates\}$

until $l = l_{max}$ or there is a clause $c \in Candidates$ with $Gain(c) > 0$ & $Weight(c) > \epsilon > 0$

$BestClauses \leftarrow$ {The k clauses $c \in Candidates$ with highest $Gain(c) > 0$ & $Weight(c) > \epsilon > 0$ }

$Candidates \leftarrow Candidates \setminus BestClauses$

return $BestClauses$

practice, we found that the estimates converge much faster than the algorithm specifies, so we run the convergence test of Degroot and Schervish (2002, p. 707) after every 100 samples and terminate if it succeeds. In addition, we use looser convergence criteria during candidate clause evaluation than during update with the chosen clause.

- When most clause weights do not change significantly with each run of L-BFGS, neither do most conditional log-likelihoods (CLLs) of ground predicates (log of Equation 2.6). We take advantage of this by storing the CLL of each sampled ground predicate and only recomputing it if a clause weight affecting it changes by more than some threshold δ. When a CLL changes, we subtract its old value from the total WPLL and add the new one. The computation of the gradient of the WPLL is similarly optimized.

- We use a lexicographic ordering on clauses to avoid redundant computations for clauses that are syntactically identical. (However, we do not detect clauses that are semantically equivalent but syntactically different; this is an NP-complete problem.) We also cache the new clauses created during each search and their counts, avoiding the need to recompute them in later searches.

3.6 Experiments

3.6.1 Datasets

We carried out experiments on two databases: the UW-CSE database used by Richardson and Domingos (2004), and McCallum's Cora database of computer science citations as segmented by Bilenko and Mooney (2003) (available at http://www.cs.utexas.edu/users/ml/riddle/data/cora.tar.gz).

The UW-CSE domain consists of 22 predicates and 1158 constants divided into 10 types. Types include: publication, person, course, etc. Predicates include: `Professor(person)`, `AdvisedBy(person1, person2)`, etc. Using typed variables, the total number of possible ground predicates is 4,055,575. The database contains a total of 3212 true ground atoms. We used the hand-coded knowledge base provided with it, which includes 94 formulas stating regularities like: each student has at most one advisor; if a student is an author of a paper, so is her advisor; etc. Notice that these statements are not always true, but are typically true.

The Cora dataset is a collection of 1295 different citations to 112 computer science research papers. We used the author, venue, title and year fields. The goal is to determine which pairs of citations refer to the same paper (i.e., to infer the truth values of all groundings of $\texttt{SameCitation}(\texttt{c1}, \texttt{c2})$). These values are available in the data. Additionally, we attempted to deduplicate the author, title and venue strings, and we labeled these manually. We defined predicates for each field that discretize the percentage of words that two strings have in common. For example, $\texttt{CommonWordsInTitles0}-\texttt{20}\%(\texttt{title1}, \texttt{title2})$ is true iff the two titles have 0-20% of their words in common. These predicates were always given as evidence, and we did not attempt to predict them. Using typed variables, the total number of possible ground predicates is 5,225,411. The database contained a total of 378,589 true ground atoms. A hand-crafted KB for this domain was provided by a colleague; it contains 26 clauses stating regularities like: if two citations are the same, their authors, venues, etc., are the same, and vice-versa; if two fields of the same type have many words in common, they are the same; etc.

3.6.2 Systems

We compared nine versions of MLN learning:

- **MLN(KB).** Weight learning applied to the hand-coded KB.
- **MLN(CL).** Structure learning using CLAUDIEN [18], followed by weight learning.
- **MLN(FO).** Structure learning using FOIL [80], followed by weight learning.
- **MLN(AL).** Structure learning using Aleph [98], followed by weight learning.
- **MLN(KB+CL).** Structure learning using CLAUDIEN with the KB providing the language bias as in Richardson and Domingos (2006), followed by weight learning on the output of CLAUDIEN merged with the KB (MLN(KB+CL)).
- **MSL_B.** Structure learning using our MSL algorithm with beam search, starting from an empty MLN.

- **KB+MSL_B.** Structure learning using our MSL algorithm with beam search, starting from the hand-coded KB.

- **MSL_B+KB.** Structure learning using our MSL algorithm with beam search, starting from an empty MLN, but allowing hand-coded clauses to be added during the first search step.

- **MSL_S.** Structure learning using our MSL algorithm with shortest-first search, starting from an empty MLN.

We added unit clauses to all nine systems. In addition, we compared MLN learning with three pure ILP approaches (CLAUDIEN (CL), FOIL (FO), and Aleph (AL)), a pure knowledge-based approach (the hand-coded KB (KB)), the combination of CLAUDIEN and the hand-coded KB as described above (KB+CL), and two pure probabilistic approaches (naive Bayes (NB) [25] and Bayesian networks (BN) [40]). Notice that ILP learners like FOIL and Aleph are not directly comparable with MSL (or CLAUDIEN), because they only learn to predict designated target predicates, as opposed to finding arbitrary regularities over all predicates. For an approximate comparison, we used FOIL and Aleph to learn rules with each predicate as the target in turn.

We used the algorithm of Richardson and Domingos (2006) to construct order-1 and order-2 attributes for the naive Bayes and Bayesian network learners. Order-1 attributes capture characteristics of individual constants in a predicate, and order-2 attributes model the relationships among the constants in the predicate. Since our goal is to measure predictive performance over all predicates, not just the $\texttt{AdvisedBy}(\texttt{x}, \texttt{y})$ predicate that Domingos and Richardson focused on, we learned a naive Bayes classifier and a Bayesian network for each predicate.

We used the same settings for CLAUDIEN as Richardson and Domingos, and let CLAUDIEN run for 24 hours on a Sun Blade 1000 workstation (CLAUDIEN only runs on Solaris machines). We used the default FOIL parameter settings except for the maximum number of variables per clause, which we set to 5 (UW-CSE) and 6 (Cora), and the minimum clause accuracy, which we set to 50%. For Aleph, we used all of the default settings except for the maximum clause length, which we set to 4 (UW-CSE) and 7 (Cora). The parameters used for our structure learning algorithms were as follows: $\pi = 0.01$ (UW-CSE) and 0.001 (Cora); maximum variables per clause = 5 (UW-

CSE) and 6 (Cora);[1] $\epsilon = 1$ (UW-CSE) and 0.01 (Cora); $\delta = 10^{-4}$; $s = 200$; $m = 100,000$; $l_{max} = 3$ (UW-CSE) and 7 (Cora); and $k = 10$ (UW-CSE) and 1 (Cora). L-BFGS was run with the following parameters: maximum iterations = 10,000 (tight) and 10 (loose); convergence threshold $= 10^{-5}$ (tight) and 10^{-4} (loose). The mean and variance of the Gaussian prior were set to 0 and 100, respectively, in all runs. A description of the parameters is given in Table 3.4. The parameters were set in an *ad hoc* manner, and per-fold optimization using a validation set could conceivably yield better results.

Table 3.4: Parameter description.

Parameter	Description
π	penalty per predicate of structure prior
δ	min. fractional clause weight change for CLL of ground predicate to be recomputed (Section 3.5)
ϵ	min. weight of candidate clauses (Tables 3.2 and 3.3)
s	max. number of candidate clauses extended in shortest-first search (Table 3.3)
m	max. number of candidate clauses retained in each iteration of shortest-first search
l_{max}	max. length of clauses returned by shortest-first search (Table 3.3)
k	max. number of clauses returned by shortest-first search (Table 3.3)

3.6.3 Methodology

In the UW-CSE domain, we used the same leave-one-area-out methodology as Richardson and Domingos (2006). In the Cora domain, we performed five runs with train-test splits of approximately equal size, ensuring that no true set of matching records was split between train and test sets to avoid contamination. We performed inference over each test ground atom to compute its probability of being true, using all other ground atoms as evidence (the log of this probability is the conditional log-likelihood (CLL) of the test ground atom). To evaluate the performance of each

[1] In the Cora domain, we further sped up learning by using syntactic restrictions on clauses similar to CLAUDIEN's declarative bias; details are in Appendix A.

system, we measured the average conditional log-likelihood (CLL) of the test atoms and area under the precision-recall curve (AUC). The advantage of the CLL is that it directly measures the quality of the probability estimates produced. The advantage of the AUC is that it is insensitive to the large number of true negatives (i.e., ground atoms that are false and predicted to be false). The precision-recall curve for a predicate is computed by varying the threshold CLL above which a ground atom is predicted to be true. For both CLL and AUC, the values we report are averages over all predicates (in the UW-CSE domain) or all non-evidence predicates (in the Cora domain), with all predicates weighted equally. We computed the standard deviations of the AUCs using the method of Richardson and Domingos (2006). To obtain probabilities from the ILP models and hand-coded KBs (required to compute CLLs and AUCs), we treated them as MLNs with all equal infinite weights.

3.6.4 Results

The results on the UW-CSE domain are shown in Table 3.5, and the results on Cora are shown in Table 3.6.[2] All versions of our MSL algorithm greatly outperformed using ILP systems to learn MLN structure, in both CLL and AUC, in both domains. This is consistent with our hypothesis that directly optimizing (pseudo-)likelihood when learning structure yields better models. In both domains, shortest-first search starting from an empty network (MSL_S) gave the best overall results, but was much slower than beam search (MSL_B) (see below).

The purely logical approaches (CL, FO, AL, KB and KB+CL) did quite poorly. This occurred because they assigned very low probabilities to true ground atoms whenever they were not entailed by the logical KB, and this occurred quite often. Learning weights for the hand-coded KBs was quite helpful, confirming the utility of transforming KBs into MLNs. However, MSL gave the best overall results. In the UW-CSE domain, refining the hand-coded KB (KB+MSL_B)) did not improve on learning from scratch. MSL was unable to break out of the local optimum represented by MLN(KB), leading to poor performance. This problem was overcome if we started instead from an empty KB but allowed hand-coded clauses to be added during the first step of beam search (MSL_B+KB)).

MSL also greatly outperformed the purely probabilistic approaches (NB and BN). This was

[2]We tried Aleph with many different parameter settings on Cora, but it always crashed by running out of memory.

Table 3.5: Experimental results on the UW-CSE database.

System	CLL	AUC
MSL_S	−0.061±0.004	0.533±0.003
MSL_B	−0.088±0.005	0.472±0.004
KB+MSL_B	−0.140±0.005	0.430±0.003
MSL_B+KB	−0.071±0.005	0.551±0.003
MLN(KB+CL)	−0.115±0.005	0.506±0.004
MLN(CL)	−0.151±0.005	0.306±0.001
MLN(FO)	−0.208±0.006	0.140±0.000
MLN(AL)	−0.223±0.006	0.148±0.001
MLN(KB)	−0.142±0.005	0.429±0.003
KB+CL	−0.789±0.012	0.318±0.003
CL	−0.574±0.010	0.170±0.004
FO	−0.661±0.003	0.131±0.001
AL	−0.579±0.006	0.117±0.000
KB	−0.812±0.011	0.266±0.003
NB	−0.370±0.005	0.390±0.003
BN	−0.166±0.004	0.397±0.002

Table 3.6: Experimental results on the Cora database.

System	CLL	AUC
MSL_S	-0.054 ± 0.000	0.813 ± 0.001
MSL_B	-0.058 ± 0.000	0.782 ± 0.001
KB+MSL_B	-0.055 ± 0.000	0.828 ± 0.001
MSL_B+KB	-0.058 ± 0.000	0.782 ± 0.001
MLN(KB+CL)	-0.069 ± 0.000	0.799 ± 0.001
MLN(CL)	-0.158 ± 0.001	0.148 ± 0.000
MLN(FO)	-0.213 ± 0.000	0.529 ± 0.001
MLN(KB)	-0.066 ± 0.000	0.809 ± 0.001
KB+CL	-0.191 ± 0.001	0.658 ± 0.001
CL	-0.693 ± 0.000	0.148 ± 0.000
FO	-0.717 ± 0.001	0.583 ± 0.001
KB	-0.229 ± 0.001	0.657 ± 0.001
NB	-0.411 ± 0.001	0.096 ± 0.001
BN	-0.257 ± 0.001	0.107 ± 0.000

consistent with our expectation because the data contained little conventional attribute-value data but much relational information.

In the UW-CSE domain, shortest-first search (MSL_S) without the speedups described in Section 3.5 did not finish running in 24 hours on a cluster of 15 dual-CPU 2.8 GHz Pentium 4 machines. With the speedups, it took an average of 5.3 hours. For beam search (MSL_B), the speedups reduced average running time from 13.7 hours to 8.8 hours on a standard 2.8 GHz Pentium 4 CPU. To investigate the contribution of each speed-up technique, we reran shortest-first search on one fold of the UW-CSE domain, leaving out one technique at a time. Clause and predicate sampling provide the largest speedups (six-fold and five-fold, respectively), and weight thresholding the smallest (1.025). None of the techniques adversely affect AUC, and predicate sampling is the only one that significantly reduces CLL (disabling it improves CLL from -0.059 to -0.038). Inference times were relatively short, taking a maximum of 1 minute (UW-CSE) and 12 minutes (Cora) on a standard 2.8 GHz Pentium 4 CPU.

The following are examples of (weighted) rules learned by MSL.

- $\texttt{TempAdvisedBy}(\texttt{x},\texttt{y}) \Rightarrow \texttt{HasFacultyPosition}(\texttt{y},\texttt{z})$. (If y advises someone, she is likely to hold a faculty position z).

- $\neg\texttt{AdvisedBy}(\texttt{x},\texttt{y}) \wedge \neg\texttt{TempAdvisedBy}(\texttt{x}',\texttt{y}) \Rightarrow \texttt{Student}(\texttt{y})$. (If y does not advise anyone, she is likely to be a student).

- $\texttt{TitleOfCit}(\texttt{t},\texttt{c}) \wedge \texttt{TitleOfCit}(\texttt{t}',\texttt{c}') \wedge \texttt{SameTitle}(\texttt{t},\texttt{t}') \Rightarrow \texttt{SameCitation}(\texttt{c},\texttt{c}')$. (If two citations c and $\texttt{c}'$ respectively have titles t and $\texttt{t}'$ that refer to the same title, then c and $\texttt{c}'$ refer to the same citation.)

- $\texttt{CommonWordsInVenues80}-\texttt{100}\%(\texttt{v},\texttt{v}') \Rightarrow \texttt{SameVenue}(\texttt{v},\texttt{v}')$. (If two venues have 80–100% of their words in common, then they are likely to refer to the same venue.)

In summary, both our algorithms are effective; we recommend using shortest-first search when accuracy is paramount, and beam search when speed is a concern.

3.7 Related Work

3.7.1 MLN Structure Learners

Subsequent to the creation of our MSL system, a few other MLN structure learners were proposed. We discuss each in turn.

Bottom-up Structure Learner (BUSL)

Despite what its name suggests, BUSL [66] is actually a hybrid bottom-up/top-down system with a significant top-down component. Mihalkova and Mooney (2007) proposed BUSL to (partially) circumvent the problems of top-down approaches to learning MLN formulas (e.g., MSL). Top-down approaches follow a 'blind' generate-and-test strategy in which formulas are systematically generated independent of the training data, resulting in the creation of many formulas that are not supported by data. Because the space of formulas is combinatorially large, it is computationally wasteful to generate formulas with no empirical support. Further, the greedy nature of the top-down approach makes it susceptible to converging to a local optimum and hence missing potentially useful formulas.

The key insight in BUSL is to use the data as a guide for the creation of candidate formulas. To do so, BUSL makes use of *relational pathfinding* [84]. Relational pathfinding finds paths of true ground atoms that are linked via their arguments, and generalizes them into first-order rules. Since each path is supported by a conjunction in the data, it focuses the search on promising regions of the space of rules. However, relational pathfinding amounts to exhaustive search over an exponential number of paths.

Hence, for tractability, BUSL restricts itself to finding very short paths (length 2) in training data. BUSL variabilizes each ground atom in the path (i.e., it replaces the constants in the atom's arguments with variables) and constructs a Markov network whose nodes are the paths viewed as Boolean variables (conjunctions of atoms).

After that, it uses the Grow-Shrink Markov network learning algorithm [12] to find the edges between the nodes in a greedy top-down manner. For each node, BUSL finds nodes connected to it by greedily adding and removing nodes from its Markov blanket using the χ^2 measure of dependence. From the maximal cliques thus created in the Markov network, BUSL creates clauses.

For each clique, it forms disjunctions of the atoms in the clique's nodes and creates clauses with all possible negation/non-negation combinations of the atoms.

BUSL computes the WPLL of the clauses and greedily adds them one at a time (in order of decreasing WPLL) to an MLN initially containing only unit clauses. After adding a clause, the weights of all clauses in the MLN are relearned to compute the new WPLL. The clause is retained in the MLN only if it increases the overall WPLL.

In Chapter 6 and 7, we respectively present the LHL and LSM systems that use relational pathfinding to a fuller extent than BUSL and outperform it.

Discriminative MLN Structure Learner

Huynh and Mooney (2008) proposed a discriminative structure learning algorithm for MLNs. It learn clauses that predict a single target predicate, unlike the MLN structure learners we present in this thesis, which model the full joint distribution of all predicates. In addition, they constrain the form of their clauses (as described below), a restriction that our structure learners do not impose.

Huynh and Mooney recognize that the ideal approach to learning discriminative clauses is to optimize the accuracy of a complete MLN. However, this would require evaluating a combinatorially large number of potential clause refinements that can be made to an existing MLN, a process that requires relearning the weights of all refined clauses and performing expensive probabilistic inference. Thus, they adopt a heuristic approach of using an off-the-shelf ILP system (Aleph [98]) to learn candidate clauses. This heuristic allows each clause to be evaluated in isolation based on the accuracy of its logical inferences. However, since the logical accuracy of a clause is only a rough guide on its contribution to the final probability model of an MLN, they generate a large number of candidate clauses and use weight learning to select among them.

All candidate clauses are added to an MLN. After that, they impose an L_1 prior on the weight of each candidate clause, and use a quasi-newton method (the Orthant-Wise limited-memory algorithm [2]) to find the optimal weights maximizing the likelihood of training data. The key feature of an L_1 prior is its tendency to force parameters to zero by strongly penalizing small weights [55]. In this manner, many candidate clauses with zero-weights are discarded, and the non-zero-weight clauses constitute the final MLN.

However, when the Orthant-Wise algorithm finds optimal weights, it has to compute the likelihood of training data (Equation 2.3), and its gradient (Equation 2.4), both of which are intractable (as described in Section 2.3). Hence, Huynh and Mooney restrict the clauses to be non-recursive definite clauses (i.e., the target predicate must appear exactly once in each clause and as its head, and all predicates in the body are evidence). This restriction causes each grounded target predicate to be independent of each another because their Markov blankets consist only of ground evidence atoms. Thus, With this restriction, the likelihood and its gradient becomes equal to the pseudo-likelihood (Equation 2.5) and its gradient (Equation 2.7) which are computationally tractable.

Iterated Local Search (ILS)

Biba et al. (2008a, 2008b) proposed the top-down ILS system that uses a stochastic step to avoid local optima. ILS begins by *randomly* selecting a clause in the current MLN to modify. (Contrast this with MSL, which always selects the highest-scoring clauses to be modified in each iteration.) Next ILS greedily modifies the clause with search operators (viz., add a literal, remove a literal, and flip the sign of a literal), always choosing the one that best improves the MLN's WPLL, until no improvement can be made. When none of the operators gives an improvement at the first greedy step, ILS chooses the one that gives the smallest WPLL decrease, so as to avoid reaching a local optimum too early. The resulting clause is then added to the MLN. The algorithm iterates the process described above until no clause is found to improve the MLN's WPLL.

The randomness in ILS helps it to overcome one of the two drawbacks of top-down approaches, i.e., local optima. However, it does not solve the other drawback of searching a large space of clauses. In fact, it exacerbates this problem by its (partially) random exploration. In Chapter 6 and 7, we present solutions that address *both* problems.

3.7.2 Non-MLN Structure Learners

We now discuss non-MLN structure learners that combine probability and logic.

MACCENT [20] is an early example of a system that combines ILP with probability. It finds the maximum entropy distribution over a set of classes consistent with a set of constraints expressed as clauses. Like MSL, it builds upon ideas on feature induction in Markov networks; however, it only

performs classification, while our goal is to do general probability estimation (i.e., learn the joint distribution of all predicates). Also, MACCENT only allows deterministic background knowledge, while our MLN structure learners allow it to be uncertain; and MACCENT classifies each example separately, while our learners allow for collective classification.

SAYU [16] combines an off-the-shelf ILP system (Aleph [98]) with tree-augmented naive Bayes (TAN [33]). Aleph is used to learn definite clauses that predict a target predicate. Each time Aleph creates a clause, SAYU adds it as a binary feature to the training data, and induces a TAN network that incorporates the new feature. The feature is retained if the TAN network increases the area under the precision-recall curve on a held-out set of data. Otherwise, SAYU reverts to the previous network. SAYU iterates the process described above until no new clauses are generated by Aleph or after a time bound. nFOIL [52] and TFOIL [53] are similar to SAYU except that they both use FOIL [80] as the ILP system to generate rules rather than Aleph. In addition, nFOIL uses naive Bayes as the statistical model rather than TAN. Note that all these systems learn clauses that predict a single target predicate, unlike the MLN structure learners we present in this thesis, which model the full joint distribution of all predicates.

3.8 Conclusion

In this chapter, we introduced the MSL algorithm for automatically learning MLN clauses and their weights. MSL explores the space of clauses in a top-down, greedy manner, guided by a weighted pseudo-likelihood measure. We showed that MSL outperformed the previous approach of Richardson and Domingos (2006), as well as purely probabilistic, purely ILP and purely knowledge-based approaches. However, the greedy nature of MSL makes it susceptible to local optima, and its generate-and-test approach of creating candidates explores a large space of clauses, many of which are not supported by data. We present algorithms for overcoming these drawbacks in Chapter 6 and 7. Before that, we turn our attention to the problem of statistical predicate invention, which is used as a component of our solution in Chapter 6.

Chapter 4

STATISTICAL PREDICATE INVENTION

4.1 Introduction

In the past few years, the statistical relational learning (SRL) community has recognized the importance of combining the strengths of statistical learning and relational learning, and developed several novel representations, as well as algorithms to learn their parameters and structure [36]. However, the problem of statistical predicate invention (SPI) has so far received little attention in the community. SPI is the discovery of new concepts, properties and relations from data, expressed in terms of the observable ones, using statistical techniques to guide the process and explicitly representing the uncertainty in the discovered predicates. These can in turn be used as a basis for discovering new predicates, which is potentially much more powerful than learning based on a fixed set of simple primitives. Essentially all the concepts used by humans can be viewed as invented predicates, with many levels of discovery between them and the sensory percepts they are ultimately based on.

In statistical learning, this problem is known as hidden or latent variable discovery, and in relational learning as predicate invention. Both hidden variable discovery and predicate invention are considered quite important in their respective communities, but are also very difficult, with limited progress to date.

One might question the need for SPI, arguing that structure learning is sufficient. Such a question can also be directed at hidden variable discovery and predicate invention, and their benefits, as articulated by their respective communities, also apply to SPI. SPI produces more compact and comprehensible models than pure structure learning, and may also improve accuracy by representing unobserved aspects of the domain. Instead of directly modeling dependencies among observed predicates, which potentially requires an exponential number of parameters, we can invent a predicate and model the dependence between it and each of the observed predicates, requiring only a linear number of parameters and reducing the risk of overfitting. In turn, invented predicates can be used to learn new formulas, allowing larger search steps, and potentially enabling us to learn more

complex models accurately.

Among the prominent approaches in statistical learning is a series of algorithms developed by Elidan, Friedman and coworkers for finding hidden variables in Bayesian networks. Elidan et al. (2001) look for structural patterns in the network that suggest the presence of hidden variables. Elidan and Friedman (2005) group observed variables by their mutual information, and create a hidden variable for each group. Central to both approaches is some form of EM algorithm that iteratively creates hidden variables, hypothesizes their values, and learns the parameters of the resulting Bayesian network. A weakness of such statistical approaches is they assume that the data is independently and identically distributed, which is not true in many real-world applications.

In relational learning, the problem is known as predicate invention (see Kramer (1995) for a survey). Predicates are invented to compress a first-order theory, or to facilitate the learning of first-order formulas. Relational learning employs several techniques for predicate invention. Predicates can be invented by analyzing first-order formulas, and forming a predicate to represent either their commonalities (interconstruction [102]) or their differences (intraconstruction [70]). A weakness of inter/intraconstruction is that they are prone to over-generating predicates, many of which are not useful. Predicates can also be invented by instantiating second-order templates [95], or to represent exceptions to learned rules [99]. Relational predicate invention approaches suffer from a limited ability to handle noisy data.

Only a few approaches to date combine elements of statistical and relational learning.

Popescul and Ungar (2004) apply k-means clustering to objects as a pre-processing step. Then they create predicates to represent the clusters and find SQL rules that relate the predicates. Wolfe and Jensen (2004) find overlapping clusters of a single type of objects. The Latent Group Model [72] uses spectral clustering to partition a graph in which nodes represent objects and edges represent relations between objects. Each partition of objects corresponds to an invented predicate. The DERL system [106] uses multinomial mixtures within entity-relational models to create a clustering structure for attributes. Long et al. [58] formulate the clustering of objects and attributes as a spectral clustering problem involving arity-2 relations. Roy et al. [89] learn a single hierarchical clustering of objects using information in relations and attributes. These previous approaches only cluster objects or attributes (a cluster corresponds to an invented unary predicate), but not relations. We would like an SPI system to automatically invent predicates that correspond to clusters of relations

as well as objects and attributes.

The FOIL-PILFS system [14] is a learning mechanism for hypertext domains. In FOIL-PILFS, class predictions produced by a naive Bayes classifier [26] are added to an ILP system (FOIL [80]) as invented predicates. FOIL-PILFS starts with a clause containing only a target predicate in its head. It then incrementally specializes the clause by adding literals to its body, so as to improve the predictive accuracy of the clause. At each stage of growing the clause, it learns a naive Bayes (NB) classifier for each variable x in the clause that can be grounded as a webpage. The positive examples of x are webpages that appear as arguments in the positive examples of the target predicate. Features of the NB classifier are the top K words with the highest mutual information with the examples of x. Using the examples of x and the K features, FOIL-PILFS trains the NB classifier, and creates a predicate that is true if the NB classifier outputs a value of 1, and is false otherwise. FOIL-PILFS stores the new predicate and uses it to extend clauses like an ordinary predicate. In its experiments, the invented predicate improves the predictive performance of FOIL-PILFS.

The SAYU-VISTA system [17] invents predicates which are added as binary features to a tree augmented naive Bayes classifer (TAN) [33]. In a TAN network, each attribute of the class variable can have at most one edge pointing to it from another attribute, in addition to the edge from the class variable. A TAN model can be constructed in polynomial time (quadratic in the number of attributes, and linear in the number of training examples). It is also guaranteed to maximize the likelihood of the training data. In SAYU-VISTA, a user has to specify a target predicate, and the types that are allowed to appear in an invented predicate. The user also has to create *linkages*, which are rules specifying how the arguments in an invented predicate are to be linked to the arguments in the target predicate. SAYU-VISTA begins by randomly selecting the arity and argument types of an invented predicate (limited to a maximum value). After that, it creates a clause that has the invented predicate as the head and an empty body. Next it performs breadth-first search to add literals to the body. For the clause that is created in each step, SAYU-VISTA links the head (i.e., the invented predicate) to the target predicate with the linkages so as to establish the dependency between the two. It then transforms the clause into a binary feature that is added to a TAN network. The search terminates when any of the following three conditions occurs: it finds a clause that meets some improvement threshold; it fully explores the search space; or it exceeds the maximum number of candidate clauses that can be created. SAYU-VISTA then restarts to find another clause. Both

FOIL-PILF and SAYU-VISTA predict only a single target predicate. Ideally, we would like an SPI system to find arbitrary regularities over all predicates.

The state-of-the-art is the infinite relational model (IRM) [43], which simultaneously clusters objects, attributes and relations. The objects can be of more than one type, and the relations can take on any number of arguments. The number of clusters for each type also need not be specified in advance. The IRM defines a generative model for the predicates and cluster assignments. It assumes that the predicates are conditionally independent given the cluster assignments, and the cluster assignments for each type are independent. IRM uses a Chinese restaurant process prior (CRP) [76] on the cluster assignments. Under the CRP, each new object is assigned to an existing cluster with probability proportional to the cluster size. Because the CRP has the property of exchangeability, the order in which objects arrive does not affect the outcome. IRM assumes that the probability p of an atom being true conditioned on cluster membership is generated according to a symmetric Beta distribution, and that the truth values of atoms are then generated according to a Bernoulli distribution with parameter p. IRM uses a top-down greedy search to find the MAP cluster assignment. It begins by assigning all object, attribute and relation symbols of the same type to a single cluster. Its search operators are: merge two clusters, split a cluster, and move an object, attribute or relation symbol from one cluster to another. Only a random subset of the possible splits is tried at each step. When it reaches a local optimum, IRM restarts. The number of restarts is limited by some user-specified maximum. As it searches for the MAP cluster assignment, it also searches for the optimal values of its CRP and Beta parameters. (Xu et al. [105] propose a closely related model.) The IRM only finds a single clustering of predicates, attributes and objects. We would like an SPI system to find *multiple* clusterings of predicates, attributes and objects, rather than just a single clustering.

In this chapter, we present MRC, an algorithm based on Markov logic [86], as a first step towards a general framework for SPI. MRC automatically invents predicates by clustering objects, attributes and relations. The invented predicates capture arbitrary regularities over all relations, and are not just used to predict a designated target relation. MRC also learns multiple clusterings, rather than just one, to represent the complexities in relational data. MRC is short for Multiple Relational Clusterings.

We describe our model in detail in the next section. In Section 4.3, we report our experiments comparing our model with IRM and the Markov logic Structure Learner (MSL; Chapter 3).

4.2 Multiple Relational Clusterings

Predicate invention is the creation of new symbols, together with formulas that define them in terms of the symbols in the data. (In a slight abuse of language, we use "predicate invention" to refer to the creation of both new predicate symbols and new constant symbols.) In this section we propose a statistical approach to predicate invention based on Markov logic. The simplest instance of statistical predicate invention is clustering, with each cluster being an invented unary predicate. More generally, all latent variables in i.i.d. statistical models can be viewed as invented unary predicates. Our goal in this paper is to extend this to relational domains, where predicates can have arbitrary arity, objects can be of multiple types, and data is non-i.i.d.

We call our approach MRC, for Multiple Relational Clusterings. MRC is based on the observation that, in relational domains, multiple clusterings are necessary to fully capture the interactions between objects. Consider the following simple example. People have coworkers, friends, technical skills and hobbies. A person's technical skills are best predicted by her coworkers's skills, and her hobbies by her friends' hobbies. If we form a single clustering of people, coworkers and friends will be mixed, and our ability to predict both skills and hobbies will be hurt. Instead, we should cluster together people who work together, and simultaneously cluster people who are friends with each other. Each person thus belongs to both a "work cluster" and a "friendship cluster." (See Figure 4.1.) Membership in a work cluster is highly predictive of technical skills, and membership in a friendship cluster is highly predictive of hobbies. The remainder of this section presents a formalization of this idea and an efficient algorithm to implement it.

Notice that multiple clusterings may also be useful in propositional domains, but the need for them there is less acute, because objects tend to have many fewer properties than relations. (For example, $\texttt{Friends}(\texttt{Anna}, \texttt{x})$ can have as many groundings as there are people in the world, and different friendships may be best predicted by different clusters Anna belongs to.)

We define our model using finite second-order Markov logic, in which variables can range over relations (predicates) as well as objects (constants). Extending Markov logic to second order involves simply grounding atoms with all possible predicate symbols as well as all constant symbols, and allows us to represent some models much more compactly than first-order Markov logic. We use it to specify how predicate symbols are clustered.

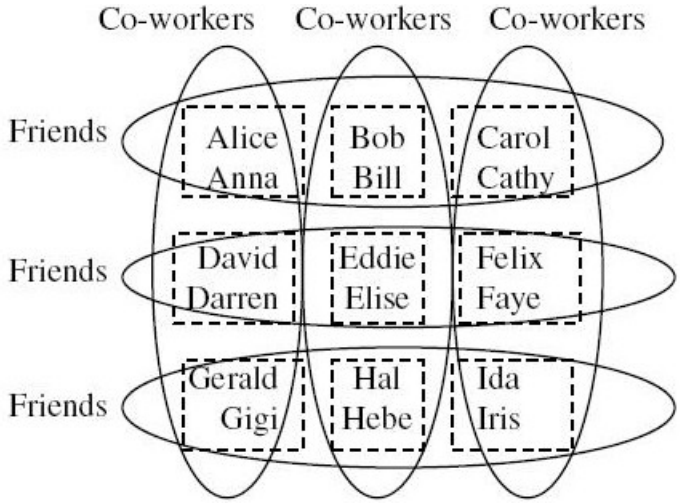

Figure 4.1: Example of multiple clusterings. Friends are clustered together in horizontal ovals, and co-workers are clustered in vertical ovals.

We use the variable r to range over predicate symbols, x_i for the ith argument of a predicate, γ_i for a cluster of ith arguments of a predicate (i.e., a set of symbols), and Γ for a clustering (i.e., a set of clusters or, equivalently, a partitioning of a set of symbols). A *cluster assignment* $\{\Gamma\}$ is an assignment of truth values to all $r \in \gamma_r$ and $x_i \in \gamma_i$ atoms.

Learning in MRC consists of finding the cluster assignment that maximizes the posterior probability $P(\{\Gamma\}|R) \propto P(\{\Gamma\}, R) = P(\{\Gamma\})P(R|\{\Gamma\})$ where R is a the vector of truth assignments to the observable ground atoms.

For simplicity, we present our rules in generic form for predicates of all arities and argument types, with n representing the arity of r; in reality, if a rule involves quantification over predicate variables, a separate version of the rule is required for each arity and argument type. We define one MLN for the prior $P(\{\Gamma\})$ component and one MLN for the likelihood $P(R|\{\Gamma\})$ component of the posterior probability with five simple rules.

The MLN for the prior component consists of four rules. The first rule states that each symbol belongs to at least one cluster:

$$\forall x \; \exists \gamma \; x \in \gamma$$

This rule is hard, i.e., it has infinite weight and cannot be violated.

The second rule states that a symbol cannot belong to more than one cluster in the same clustering:

$$\forall x, \gamma, \gamma', \Gamma \;\; x \in \gamma \wedge \gamma \in \Gamma \wedge \gamma' \in \Gamma \wedge \gamma \neq \gamma' \Rightarrow x \notin \gamma'$$

This rule is also hard.

If we say that $r(x_1, \ldots, x_n)$ is in the *combination of clusters* $(\gamma_r, \gamma_1, \ldots, \gamma_n)$, then r is in cluster γ_r and x_i is in cluster γ_i. The third rule says that each atom appears in exactly one combination of clusters and is also hard:

$$\forall r, x_1, \ldots, x_n \; \exists^1(\gamma_r, \gamma_1, \ldots, \gamma_n) \;\; r(x_1, \ldots, x_n) \in (\gamma_r, \gamma_1, \ldots, \gamma_n)$$

To combat the proliferation of clusters and consequent overfitting, we impose an exponential prior on the number of clusters, represented by the fourth rule

$$\forall \gamma \exists x \; x \in \gamma$$

with negative weight $-\lambda$. The parameter λ is fixed during learning, and is the penalty in log-posterior incurred by adding a cluster to the model. Thus larger λs lead to fewer clusterings being formed.[1] Note that $P(\{\Gamma\}) = 0$ for any $\{\Gamma\}$ that violates any of the above hard rules. For the remainder, $P(\{\Gamma\})$ reduces to the exponential prior.

The MLN for the likelihood component contains the key rule in the model, which states that the truth value of an atom is determined by the cluster combination it belongs to:

$$\forall r, x_1, \ldots, x_n, +\gamma_r, +\gamma_1, \ldots, +\gamma_n \;\; r \in \gamma_r \wedge x_1 \in \gamma_1 \wedge \ldots \wedge x_n \in \gamma_n \Rightarrow r(x_1, \ldots, x_n)$$

This rule is soft. The "+" notation is syntactic sugar that signifies the MLN contains an instance of this rule *with a separate weight* for each tuple of clusters $(\gamma_r, \gamma_1, \ldots, \gamma_n)$. As we will see below, this weight is the log-odds that a random atom in this cluster combination is true. Thus, this is the rule that allows us to predict the probability of query atoms given the cluster memberships of the symbols in them. We call this the *atom prediction* rule.

It is easily seen that, given a cluster assignment, the likelihood MLN decomposes into a separate MLN for each combination of clusters, and the weight of the corresponding atom prediction rule

[1] We have also experimented with using a Chinese restaurant process prior (CRP, Pitman (2002)), and the results were similar. We thus use the simpler exponential prior.

is the log odds of an atom in that combination of clusters being true. (Recall that, by design, each atom appears in exactly one combination of clusters.) Further, given a cluster assignment, atoms with unknown truth values do not affect the estimation of weights, because they are graph-separated from all other atoms by the cluster assignment. If t_k is the empirical number of true atoms in cluster combination k, and f_k the number of false atoms, we estimate w_k as $\log((t_k + \beta)/(f_k + \beta))$, where β is a smoothing parameter.

Conversely, given the model weights, we can use inference to assign probabilities of membership in combinations of clusters to all atoms. Thus the learning problem can in principle be solved using an EM algorithm, with cluster assignment as the E step, and MAP estimation of weights as the M step. However, while the M step in this algorithm is trivial, the E step is extremely complex. We begin by simplifying the problem by performing hard assignment of symbols to clusters (i.e., instead of computing probabilities of cluster membership, a symbol is simply assigned to its most likely cluster). Since performing an exhaustive search over cluster assignments is infeasible, the key is to develop an intelligent tractable approach. Since, given a cluster assignment, the MAP weights and thus the posterior probability can be computed in closed form, a better alternative to EM is simply to search over cluster assignments, evaluating each assignment by its posterior probability. This can be viewed as a form of structure learning, where a structure is a cluster assignment.

Table 4.1 shows the pseudocode for our MRC learning algorithm. The basic idea is the following: when clustering sets of symbols related by atoms, each refinement of one set of symbols potentially forms a basis for the further refinement of the related clusters. MRC is thus composed of two levels of search: the top level finds clusterings, and the bottom level finds clusters. At the top level, MRC is a recursive procedure whose inputs are a cluster of predicates γ_r per arity and argument type, and a cluster of symbols γ_i per type. In the initial call to MRC, each γ_r is the set of all predicate symbols with the same number and type of arguments, and γ_i is the set of all constant symbols of the ith type. At each step, MRC creates a cluster symbol for each cluster of predicate and constant symbols it receives as input. Next it clusters the predicate and constant symbols, creating and deleting cluster symbols as it creates and destroys clusters. It then calls itself recursively with each possible combination of the clusters it formed. For example, suppose the data consists of binary predicates $r(x_1, x_2)$, where x_1 and x_2 are of different type. If r is clustered into γ_r^1 and γ_r^2, x_1 into x_1^1 and x_1^2, and x_2 into x_2^1 and x_2^2, MRC calls itself recursively with the cluster combinations

$(\gamma_r^1, \gamma_1^1, \gamma_2^1)$, $(\gamma_r^1, \gamma_1^1, \gamma_2^2)$, $(\gamma_r^1, \gamma_1^2, \gamma_2^1)$, $(\gamma_r^1, \gamma_1^2, \gamma_2^2)$, $(\gamma_r^2, \gamma_1^1, \gamma_2^1)$, etc.

Within each recursive call, MRC uses greedy search with restarts to find the MAP clustering of the subset of predicate and constant symbols it received. It begins by assigning all constant symbols of the same type to a single cluster, and similarly for predicate symbols of the same arity and argument type. The search operators used are: move a symbol between clusters, merge two clusters, and split a cluster. (If clusters are large, only a random subset of the splits is tried at each step.) A greedy search ends when no operator increases posterior probability. Restarts are performed, and they give different results because of the random split operator used.

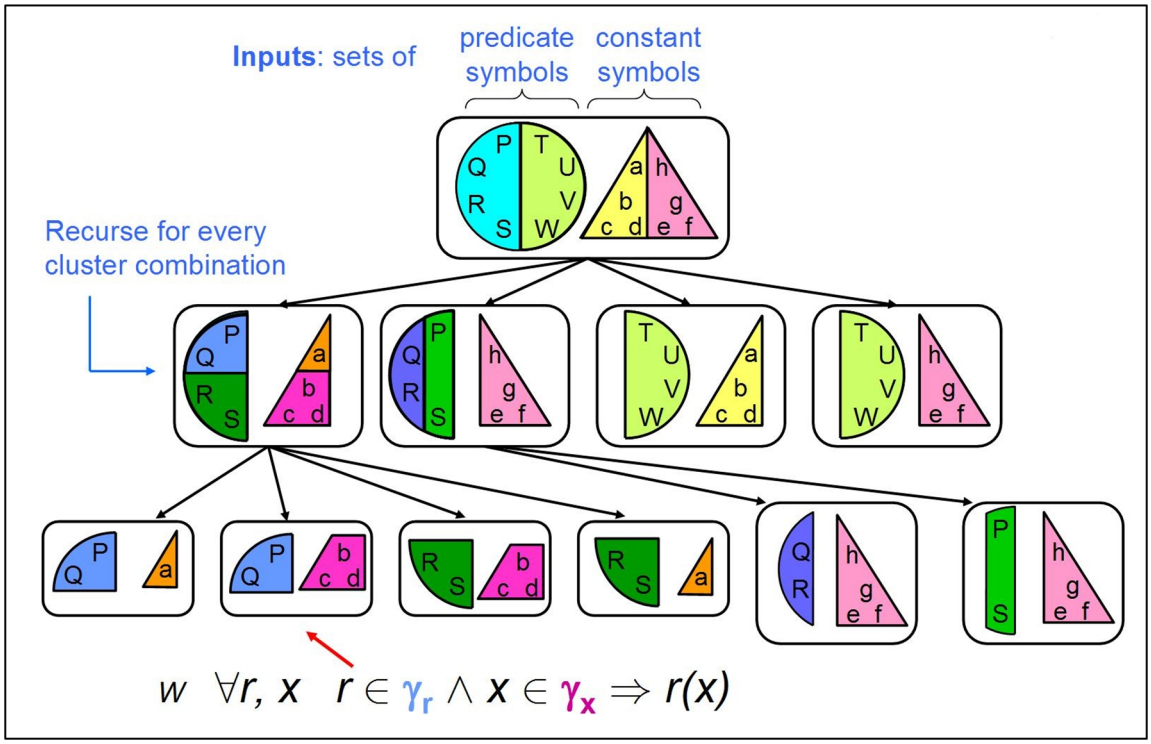

Figure 4.2: Illustration of MRC algorithm.

Figure 4.2 provides an illustration of the MRC algorithm. At the top of the figure, MRC is initially provided with a cluster of predicate symbols (P-W) and constant symbols (a-h). It then finds a clustering of the predicate symbols ({P,Q,R,S},{T,U,V,W}) and constant symbols ({a,b,c,d}, {e,f,g,h}). After that, it recurse by using each combination of predicate cluster and constant cluster as input. Since each initial set of symbols is split into two clusters, MRC is called recursively four

times at the second level of the figure, each time with a different combination of predicate cluster and constant cluster. MRC continues to call itself recursively until the posterior probability does not improve by further refining its input clusters. This occurs at the bottom level of the figure. The figure shows an atom prediction rule with weight w that corresponds to a cluster combination at which MRC bottoms out. γ_r in the rule corresponds to the blue cluster $\{$Q,P$\}$, and γ_x corresponds to the pink cluster $\{$b,c,d$\}$. Note that the blue and pink clusters do not exist *a priori* in the domain and are created by MRC, i.e., MRC discovers their *latent* structure in the data. Thus, we can interpret MRC as finding *latent* atom prediction rules. Also note that symbol P belongs to different clusters in the leftmost and rightmost cluster combinations at the bottom of the figure

Notice that in each call of MRC, it forms a clustering for each of its input clusters, thereby always satisfying the first two hard rules in the MLN. MRC also always satisfies the third hard rule because it only passes the atoms in the current combination to each recursive call.

MRC terminates when no further refinement increases posterior probability and returns the finest clusterings produced. In other words, if we view MRC as growing a tree of clusterings, it returns the leaves. Conceivably, it might be useful to retain the whole tree, and perform shrinkage [63] over it. This is an item for future work. Notice that the clusters created at a higher level of recursion constrain the clusters that can be created at lower levels, e.g., if two symbols are assigned to different clusters at a higher level, they cannot be assigned to the same cluster in subsequent levels. Notice also that predicate symbols of different arities and argument types are never clustered together. This is a limitation that we plan to overcome in the future.

4.3 Experiments

In our experiments, we compare MRC with the Infinite Relational Model (IRM; described in Section 4.1) and the Markov logic Structure Learner (MSL; Chapter 3). We use the beam search version of MSL that is implemented in the open-source Alchemy package [50].

4.3.1 Datasets

We compared MRC to IRM and MSL on all four datasets used by Kemp et al. (2006) .[2]

[2]The IRM code and datasets are publicly available at http://www.psy.cmu.edu/~ckemp/code/irm.html.

Table 4.1: MRC algorithm.

function *MRC*(C, R)

inputs: $C = (\gamma_{r_1}, \ldots, \gamma_{r_m}, \gamma_1, \ldots, \gamma_n)$, a combination of clusters, where γ_{r_i} is a cluster of relation symbols with the same number and type of arguments, and γ_j is a cluster of constant symbols of the same type

R, ground atoms formed from the symbols in C

output: $D = \{(\gamma'_{r_1}, \ldots, \gamma'_{r_m}, \gamma'_1, \ldots, \gamma'_n)\}$, a set of cluster combinations where $\gamma'_i \subseteq \gamma_i$

note: Γ_i is a clustering of the symbols in γ_i, i.e., $\Gamma_i = \{\gamma_i^1, \ldots, \gamma_i^k\}$, $\gamma_i^j \subseteq \gamma_i, \bigcup_{j=1}^{k} \gamma_i^j = \gamma_i$, and $\gamma_i^j \cap \gamma_i^k = \emptyset, j \neq k$. $\{\Gamma_i\}$ is a set of clusterings.

$\Gamma_i \leftarrow \{\gamma_i\}$ for all γ_i in C

$\{\Gamma_i^{Best}\} \leftarrow \{\Gamma_i\}$

for $s \leftarrow 0$ to *MaxSteps* **do**

$\{\Gamma_i^{Tmp}\} \leftarrow$ best change to any clustering in $\{\Gamma_i\}$

if $P(\{\Gamma_i^{Tmp}\}|R) > P(\{\Gamma_i\}|R)$

$\{\Gamma_i\} \leftarrow \{\Gamma_i^{Tmp}\}$

if $P(\{\Gamma_i\}|R) > P(\{\Gamma_i^{Best}\}|R)$

$\{\Gamma_i^{Best}\} \leftarrow \{\Gamma_i\}$

else if for the last $MaxBad$ consecutive iterations

$P(\{\Gamma_i^{Tmp}\}|R) \leq P(\{\Gamma_i\}|R)$

reset $\Gamma_i \leftarrow \{\gamma_i\}$ for all γ_i in C

if $\Gamma_i^{Best} = \{\gamma_i\}$ for all γ_i in C

return C

$D \leftarrow \emptyset$

for each $C' \in \Gamma_{r_1}^{Best} \times \ldots \times \Gamma_{r_m}^{Best} \times \Gamma_1^{Best} \times \ldots \times \Gamma_n^{Best}$

$R' \leftarrow$ ground atoms formed from the symbols in C'

$D \leftarrow D \cup$ *MRC*(C',R')

return D

Animals. This dataset contains a set of animals and their features [74]. It consists exclusively of unary predicates of the form $\mathtt{f}(\mathtt{a})$, where $\mathtt{f}$ is a feature and $\mathtt{a}$ is an animal (e.g., $\mathtt{Swims}(\mathtt{Dolphin})$). There are 50 animals, 85 features, and thus a total of 4250 ground atoms, of which 1562 are true. This is a simple propositional dataset with no relational structure, but it is useful as a "base case" for comparison. Notice that, unlike traditional clustering algorithms, which only cluster objects by features, MRC and IRM also cluster features by objects. This is known as bi-clustering or co-clustering, and has received considerable attention in the recent literature (e.g., [23]).

UMLS. UMLS contains data from the Unified Medical Language System, a biomedical ontology [64]. It consists of binary predicates of the form $\mathtt{r}(\mathtt{c}, \mathtt{c}')$, where $\mathtt{c}$ and $\mathtt{c}'$ are biomedical concepts (e.g., $\mathtt{Antibiotic}$, $\mathtt{Disease}$), and $\mathtt{r}$ is a relation between them (e.g., $\mathtt{Treats}$, $\mathtt{Diagnoses}$). There are 49 relations and 135 concepts, for a total of 893,025 ground atoms, of which 6529 are true.

Kinship. This dataset contains kinship relationships among members of the Alyawarra tribe from Central Australia [22]. Predicates are of the form $\mathtt{k}(\mathtt{p}, \mathtt{p}')$, where $\mathtt{k}$ is a kinship relation and $\mathtt{p}$, $\mathtt{p}'$ are persons. There are 26 kinship terms and 104 persons, for a total of 281,216 ground atoms, of which 10,686 are true.

Nations. This dataset contains a set of relations among nations and their features [90]. It consists of binary and unary predicates. The binary predicates are of the form $\mathtt{r}(\mathtt{n}, \mathtt{n}')$, where $\mathtt{n}$, $\mathtt{n}'$ are nations, and $\mathtt{r}$ is a relation between them (e.g., $\mathtt{ExportsTo}$, $\mathtt{GivesEconomicAidTo}$). The unary predicates are of the form $\mathtt{f}(\mathtt{n})$, where $\mathtt{n}$ is a nation and $\mathtt{f}$ is a feature (e.g., $\mathtt{Communist}$, $\mathtt{Monarchy}$). There are 14 nations, 56 relations and 111 features, for a total of 12,530 ground atoms, of which 2565 are true.

4.3.2 Methodology

Experimental evaluation of statistical relational learners is complicated by the fact that in many cases the data cannot be separated into independent training and test sets. While developing a long-term solution for this remains an open problem, we used an approach that is general and robust: performing cross-validation by atom. For each dataset, we performed ten-fold cross-validation by randomly dividing the atoms into ten folds, training on nine folds at a time, and testing on the remaining one. This can be seen as evaluating the learners in a *transductive* setting, because an

object (e.g., `Leopard`) that appears in the test set (e.g., in `MeatEater(Leopard)`) may also appear in the training set (e.g., in `Quadrapedal(Leopard)`). In the training data, the truth values of the test atoms are set to `unknown`, and their actual values (`true/false`) are not available. Thus learners must perform generalization in order to be able to infer the test atoms, but the generalization is aided by the dependencies between test atoms and training ones.

Notice that MSL is not directly comparable to MRC and IRM because it makes the closed-world assumption, i.e., all atoms not in its input database are assumed to be false. Our experiments require the test atoms to be open-world. For an approximate comparison, we set all test atoms to false when running MSL. Since in each run these are only 10% of the training set, setting them to false does not greatly change the sufficient statistics (true clause counts) learning is based on. We then ran MC-SAT [77] on the MLNs learned by MSL to infer the probabilities of the test atoms. (The MSL parameters are specified in Appendix B.)

To evaluate the performance of MRC, IRM and MSL, we measured the average conditional log-likelihood of the test atoms given the observed training ones (CLL), and the area under the precision-recall curve (AUC). The advantage of the CLL is that it directly measures the quality of the probability estimates produced. The advantage of the AUC is that it is insensitive to the large number of true negatives (i.e., atoms that are false and predicted to be false). The precision-recall curve for a predicate is computed by varying the threshold CLL above which an atom is predicted to be true.

For IRM, we used all of the default settings in its publicly available software package (except that we terminated runs after a fixed time rather than a fixed number of iterations). For our model, we set both parameters λ and β to 1 (without any tuning). We ran IRM for ten hours on each fold of each dataset. We also ran MRC for ten hours per fold, on identically configured machines, for the first level of clustering. Subsequent levels of clustering were permitted 100 steps. MRC took a total of 3-10 minutes for the subsequent levels of clustering, negligible compared to the time required for the first level and by IRM. We allowed a much longer time for the first level of clustering because this is where the sets of objects, attributes and relations to be clustered are by far the largest, and finding a good initial clustering is important for the subsequent learning.

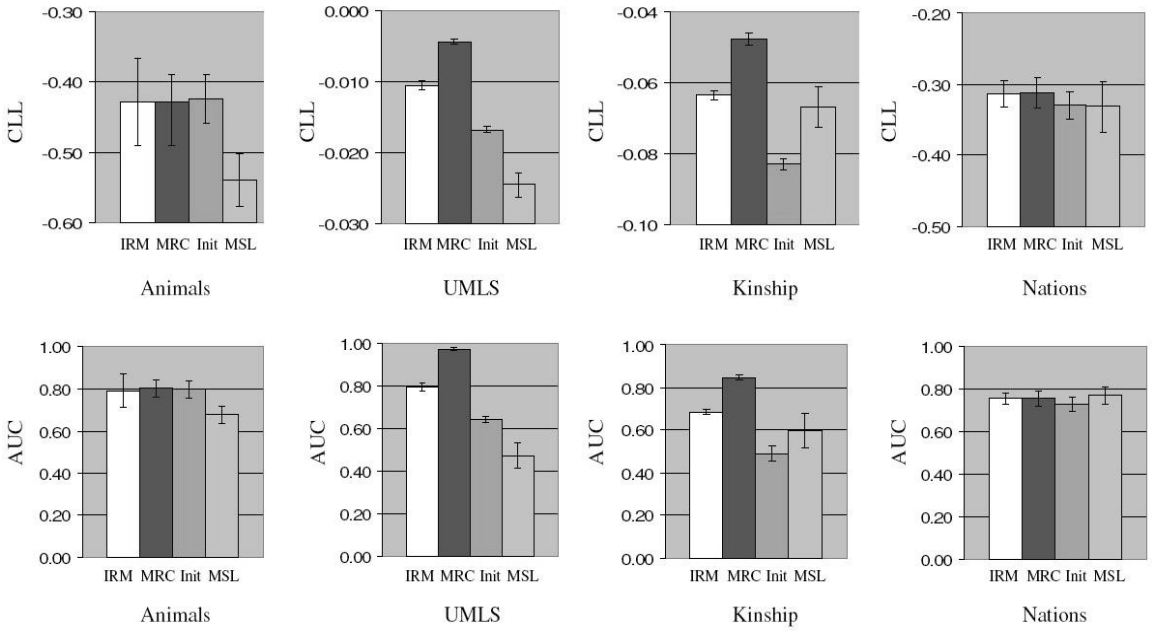

Figure 4.3: Comparison of MRC, IRM and MSL using ten-fold cross-validation: average conditional log-likelihood of test atoms (CLL) and average area under the precision-recall curve (AUC). Init is the initial clustering formed by MRC. Error bars are one standard deviation in each direction.

4.3.3 Results

Figure 4.3 reports the CLL and AUC for MRC, IRM and MSL, averaged over the ten folds of each dataset. We also report the results obtained using just the initial clustering formed by MRC (Init), in order to evaluate the usefulness of learning multiple clusterings.

MSL does worse than MRC and IRM on all datasets except Nations. On Nations, it does worse than MRC and IRM in terms of CLL, but approximately ties them in terms of AUC. Many of the relations in Nations are symmetric, e.g., if country A has a military conflict with B, then the reverse is usually true. MSL learns a rule to capture the symmetry and consequently does well in terms of AUC.

MRC outperforms IRM on UMLS and Kinship, and ties it on Animals and Nations. The difference on UMLS and Kinship is quite large. Animals is the smallest and least structured of the datasets, and it is conceivable that it has little room for improvement beyond a single clustering. The difference in performance between MRC and IRM correlates strongly with dataset size. (Notice that UMLS and Kinship are at least an order of magnitude larger than Animals and Nations.) This suggests that sophisticated algorithms for statistical predicate invention may be of most use in even larger datasets, which we plan to experiment with in the future.

MRC outperforms Init on all domains except Animals. The differences on Nations are not significant, but on UMLS and Kinship they are very large. These results show that forming multiple clusterings is key to the good performance of MRC. In fact, Init does considerably worse than IRM on UMLS and Kinship; we attribute this to the fact that IRM performs a search for optimal parameter values, while in MRC these parameters were simply set to default values without any tuning on data. This suggests that optimizing parameters in MRC could lead to further performance gains.

In the Animals dataset, MRC performs at most three levels of cluster refinement. On the other datasets, it performs about five. The average total numbers of clusters generated are: Animals, 202; UMLS, 405; Kinship, 1044; Nations, 586. The average numbers of atom prediction rules learned are: Animals, 305; UMLS, 1935; Kinship, 3568; Nations, 12,169. We provide examples of multiple clusterings that MRC learned for the UMLS dataset in Figures 4.4- 4.6.

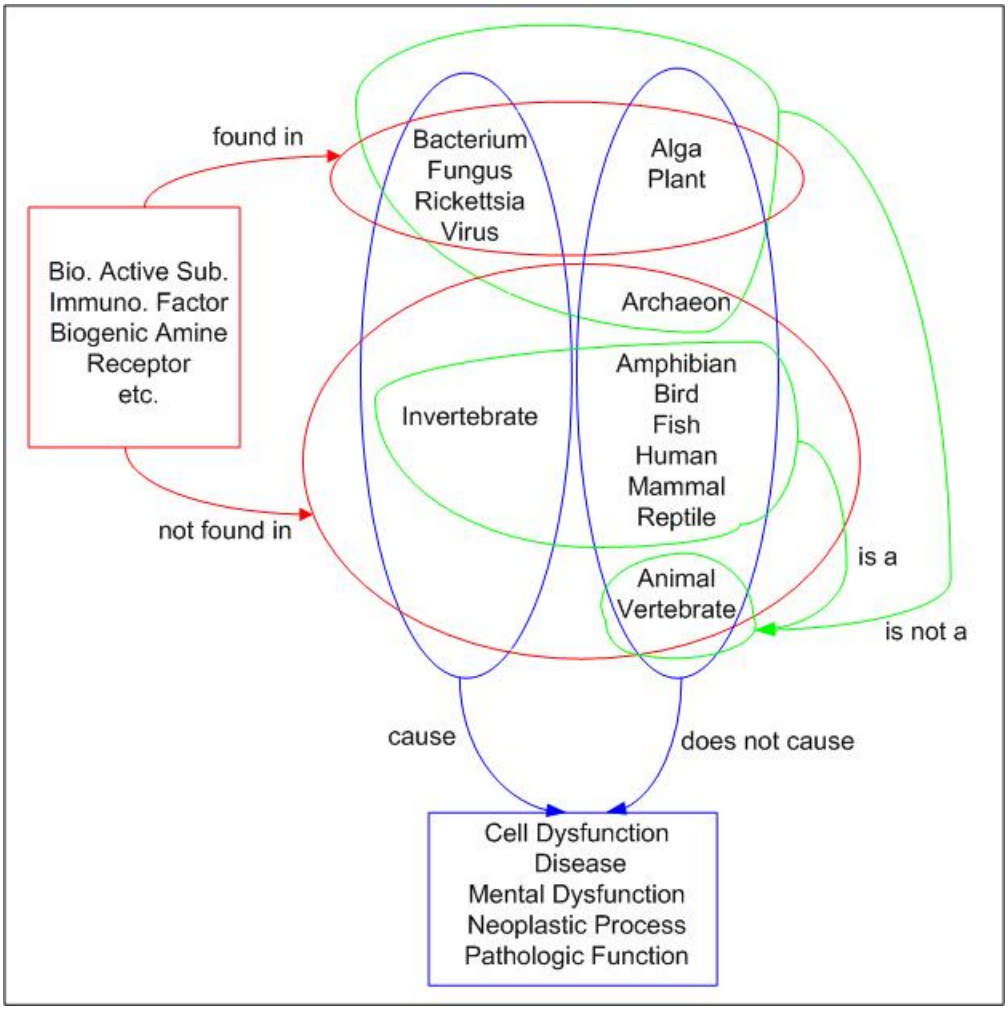

Figure 4.4: In the above figure, the organisms are clustered in three different ways according to: what are found in them (red), their pathologic properties (blue), and whether they are animals/vertebrates (green).

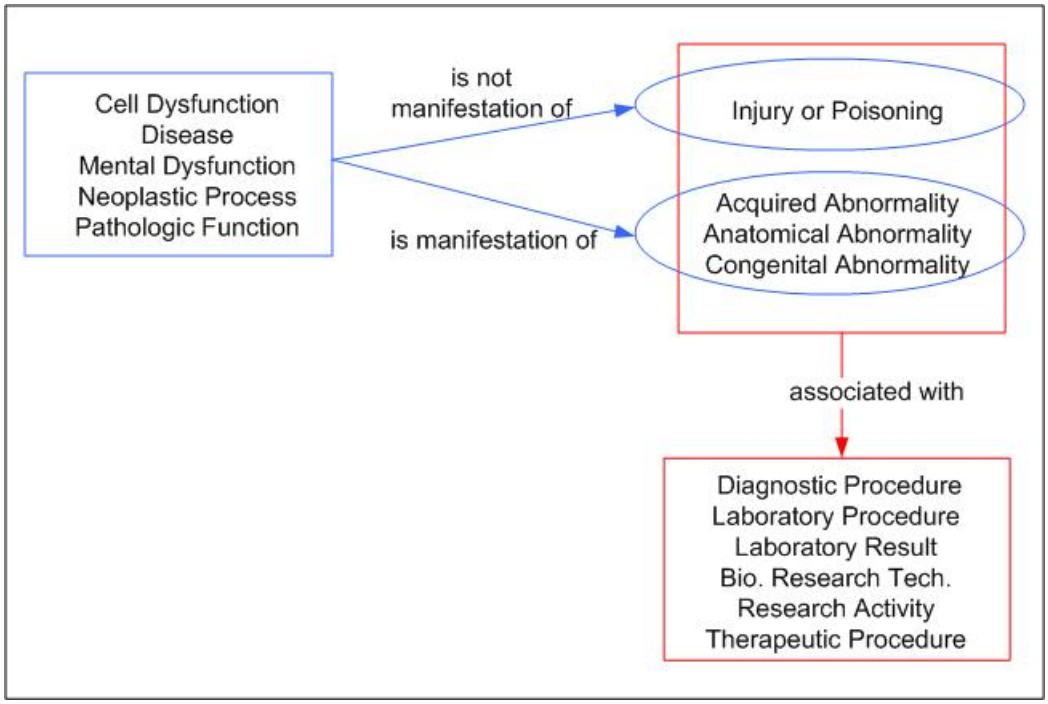

Figure 4.5: In the above figure, there are two clusterings of "Injury or Poisoning" and the Abnormalities according to what they are manifestations of (blue) and what they are associated with (red).

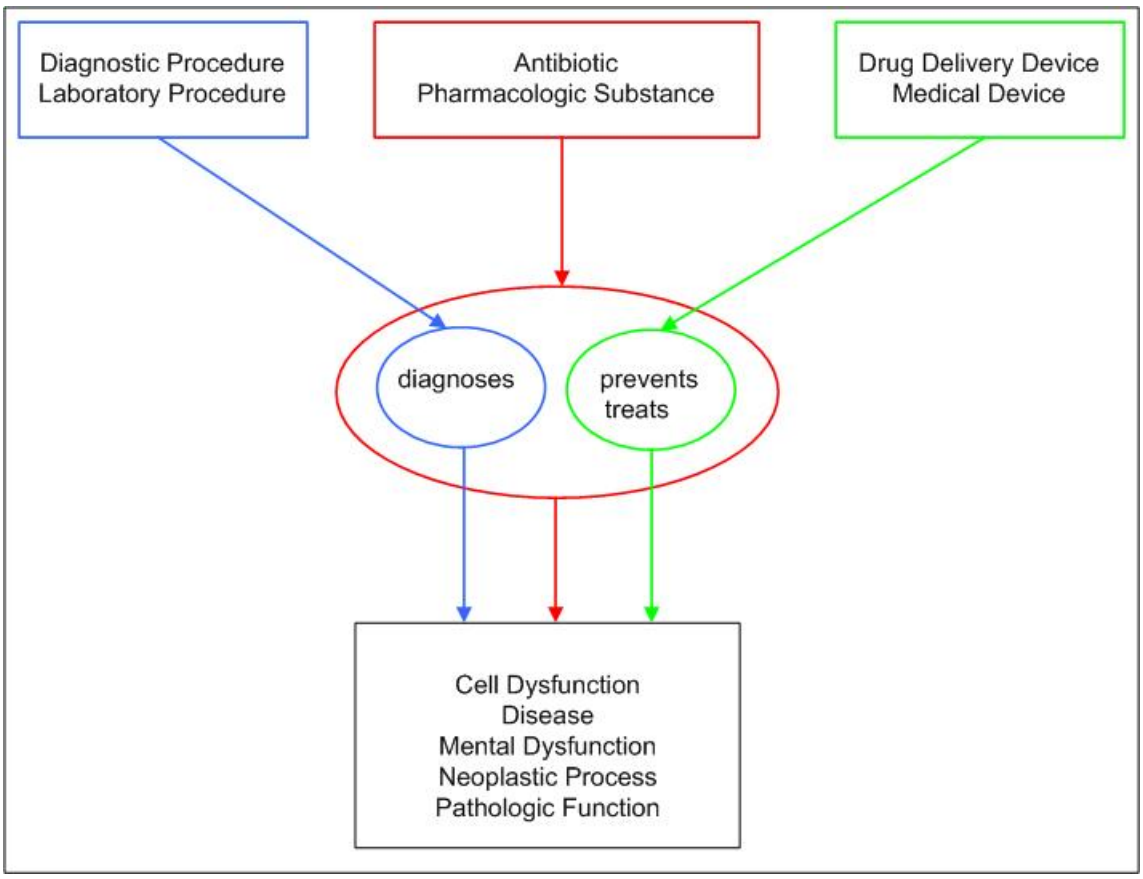

Figure 4.6: In the above figure, the relations “diagnoses”, “prevents” and “treats” are clustered in three ways. “Antibiotic” and “Pharmacologic Substance” diagnose, prevent and treat diseases (red). “Diagnostic Procedure” and “Laboratory Procedure” only diagnose but do not prevent or treat diseases (blue). “Drug Delivery Device” and “Medical Device” prevent and treat diseases but do not diagnose them (green).

4.4 Conclusion

In this chapter, we proposed statistical predicate invention (i.e., the discovery of new concepts, properties and relations in structured data) as a key problem for statistical relational learning. We then introduced MRC, an approach to SPI based on second-order Markov logic. MRC forms multiple relational clusterings of the symbols in the data and iteratively refines them. Empirical comparisons with a Markov logic structure learning system and a state-of-the-art relational clustering system on four datasets show the efficacy of our model. In the next chapter, we apply SPI to the problem of extracting knowledge in the form of semantic networks from text.

Chapter 5

EXTRACTING SEMANTIC NETWORKS FROM TEXT VIA RELATIONAL CLUSTERING

5.1 Introduction

A long-standing goal of AI is to build an autonomous agent that can read and understand text. The natural language processing (NLP) community attempted to achieve this goal in the 1970's and 1980's by building systems for understanding and answering questions about simple stories [13, 56, 92, 27]. These systems parsed text into a network of predefined concepts and created a knowledge base from which inferences can be made. However, they required a large amount of manual engineering, only worked on small text sizes and were not robust enough to perform well on unrestricted naturally occurring text. Gradually, research in this direction petered out.

Interest in the goal has been recently rekindled [67, 31] by the abundance of easily accessible Web text and by the substantial progress over the last few years in machine learning and NLP. The confluence of these three developments led to efforts to extract facts and knowledge bases from the Web [15]. Two recent steps in this direction are a system by Pasca et. al (2006) and TextRunner [3]. Both systems extract facts on a large scale from Web corpora in an unsupervised manner. Pasca et. al's system derives relation-specific extraction patterns from a starting set of seed facts, acquires candidate facts using the patterns, adds high-scoring facts to the seeds, and iterates until some convergence criterion. TextRunner uses a domain-independent approach to extract a large set of relational tuples of the form $r(x, y)$ where x and y are strings denoting objects, and r is a string denoting a relation between the objects. It uses a lightweight noun phrase chunker to identify objects, and heuristically determines the text between objects as relations. These are good first steps, but they still fall short of the goal. While they can quickly acquire a large database of ground facts in an unsupervised manner, they are not able to learn general knowledge that is embedded in the facts.

Another line of recent research takes the opposite approach. Semantic parsing [110, 104, 68] is

the task of mapping a natural language sentence into logical form. The logical statements constitute a knowledge base that can be used to perform some task like answering questions. Semantic parsing systems require a training corpus of sentences annotated with their associated logical forms (i.e., they are supervised). These systems are then trained to induce a parser that can convert novel sentences to their logical forms. Even though these systems can create knowledge bases directly, their need for annotated training data prevents them from scaling to large corpora like the Web.

In this chapter, we present SNE, a scalable, unsupervised and domain-independent system that simultaneously extracts high-level relations and concepts, and learns a semantic network [79] from text. It first uses TextRunner to extract ground facts as triples from text, and then extract knowledge from the triples. TextRunner's triples are noisy, sparse and contain many co-referent objects and relations. Our system has to overcome these challenges in order to extract meaningful high-level relations and concepts from the triples in an unsupervised manner. It does so with a probabilistic model that clusters objects by the objects that they are related to, and that clusters relations by the objects they relate. This allows information to propagate between clusters of relations and clusters of objects as they are created. Each cluster represents a high-level relation or concept. A concept cluster can be viewed as a node in a graph, and a relation cluster can be viewed as a link between the concept clusters that it relates. Together the concept clusters and relation clusters define a simple semantic network. Figure 5.1 illustrates part of a semantic network that our approach learns. SNE is short for Semantic Network Extractor.

SNE is based on Markov logic [86], and is related to the Multiple Relational Clusterings (MRC) model [46] described in Chapter 4. SNE is our first step towards creating a system that can extract an arbitrary semantic network directly from text. Ultimately, we want to tightly integrate the information extraction TextRunner component and the knowledge learning SNE component to form a self-contained *knowledge extraction* system. This tight integration will enable information to flow between both tasks, allowing them to be solved jointly for better performance [61].

In the next section, we describe our model in detail. After that, we report our experiments comparing our model with three alternative approaches in Section 5.3 and discuss related work in Section 5.4.

5.2 Semantic Network Extraction

SNE simultaneously clusters objects and relations in an unsupervised manner, without requiring the number of clusters to be specified in advance. The object clusters and relation clusters respectively form the nodes and links of a semantic network. A link exists between two nodes if and only if a true ground fact can be formed from the symbols in the corresponding relation and object clusters.

When faced with the task of extracting knowledge from noisy and sparse data like that used in our experiments, we have to glean every bit of useful information from the data to form coherent clusters. SNE does this by jointly clustering objects and relations. In its algorithm, SNE allows information from object clusters it has created at each step to be used in forming relation clusters, and vice versa. As we shall see later in our experimental results, this joint clustering approach does better than clustering objects and relations separately.

SNE is defined using a form of finite second-order Markov logic in which variables can range over relations (predicates) as well as objects (constants). Extending Markov logic to second order involves simply grounding atoms with all possible predicate symbols as well as all constant symbols, and allows us to represent some models much more compactly than first-order Markov logic.

For simplicity, we assume that relations are binary in our definition of SNE, i.e., relations are of the form $r(x, y)$ where r is a relation symbol, and x and y are object symbols. We use γ_i and Γ_i to respectively denote a cluster and clustering (i.e., a partitioning) of symbols of type i. If r, x and y are respectively in cluster γ_r, γ_x, and γ_y, we say that $r(x, y)$ is in the *cluster combination* $(\gamma_r, \gamma_x, \gamma_y)$.

The learning problem in SNE consists of finding the cluster assignment $\Gamma = (\Gamma_r, \Gamma_x, \Gamma_y)$ that maximizes the posterior probability $P(\Gamma|R) \propto P(\Gamma, R) = P(\Gamma)P(R|\Gamma)$, where R is a vector of truth assignments to the observable $r(x, y)$ ground atoms.

We define one MLN for the likelihood $P(R|\Gamma)$ component and one MLN for the prior $P(\Gamma)$ component of the posterior probability with just four simple rules.

The MLN for the likelihood component only contains one rule stating that the truth value of an atom is determined by the cluster combination it belongs to:

$$\forall r, x, y, +\gamma_r, +\gamma_x, +\gamma_y \quad r \in \gamma_r \wedge x \in \gamma_x \wedge y \in \gamma_y \Rightarrow r(x, y)$$

This rule is soft. The "+" notation is syntactic sugar that signifies that there is an instance of this rule *with a separate weight* for each cluster combination $(\gamma_r, \gamma_x, \gamma_y)$. This rule predicts the probability of query atoms given the cluster memberships of the symbols in them. This is known as the *atom prediction* rule. Given a cluster assignment, the MAP weight w_k of an instance of the atom prediction rule is given by $\log(t_k/f_k)$, where t_k is the empirical number of true atoms in cluster combination k, and f_k is the number of false atoms. Adding smoothing parameters α and β, we estimate the MAP weight as $\log((t_k + \alpha)/(f_k + \beta))$.

Three rules are defined in the MLN for the prior component. The first rule states that each symbol belongs to exactly one cluster:

$$\forall x \; \exists^1 \gamma \; x \in \gamma$$

This rule is hard, i.e., it has infinite weight and cannot be violated.

The second rule imposes an exponential prior on the number of cluster combinations. This rule combats the proliferation of cluster combinations and consequent overfitting, and is represented by the formula

$$\forall \gamma_r, \gamma_x, \gamma_y \; \exists r, x, y \quad r \in \gamma_r \wedge x \in \gamma_x \wedge y \in \gamma_y$$

with negative weight $-\lambda$. The parameter λ is fixed during learning and is the penalty in log-posterior incurred by adding a cluster combination to the model. Thus larger λs lead to fewer cluster combinations being formed. This rule represents the complexity of the model in terms of the number of instances of the atom prediction rule (which is equal to the number of cluster combinations).

The last rule encodes the belief that most symbols tend to be in different clusters. It is represented by the formula

$$\forall x, x', \gamma_x, \gamma_{x'} \;\; x \in \gamma_x \wedge x' \in \gamma_{x'} \wedge x \neq x' \Rightarrow \gamma_x \neq \gamma_{x'}$$

with positive weight μ. The parameter μ is also fixed during learning. We expect there to be many concepts and high-level relations in a large heterogenous body of text. The tuple extraction process samples instances of these concepts and relations sparsely, and we expect each concept or relation to have only a few instances sampled, in many cases only one. Thus we expect most pairs of symbols to be in different concept and relation clusters.

The equation for the log-posterior, as defined by the two MLNs, can be written in closed form as follows (see Appendix C for derivation):

$$\log P(\Gamma|R) = \sum_{k\in K}\left[t_k \log\left(\frac{t_k+\alpha}{t_k+f_k+\alpha+\beta}\right) + f_k \log\left(\frac{f_k+\beta}{t_k+f_k+\alpha+\beta}\right)\right] - \lambda m_{cc} + \mu d + \mathcal{C} \tag{5.1}$$

where K is the set of cluster combinations, m_{cc} is the number of cluster combinations, d is the number of pairs of symbols that belong to different clusters, and $\mathcal{C}$ is a constant.

Rewriting the equation, the log-posterior can be expressed as

$$\begin{aligned}\log P(\Gamma|R) = &\sum_{k\in K^+}\left[t_k \log\left(\frac{t_k+\alpha}{t_k+f_k+\alpha+\beta}\right) + f_k \log\left(\frac{f_k+\beta}{t_k+f_k+\alpha+\beta}\right)\right] \\ &+ \sum_{k\in K^-}\left[f_k \log\left(\frac{f_k+\beta}{t_k+f_k+\alpha+\beta}\right)\right] - \lambda m_{cc} + \mu d + \mathcal{C}\end{aligned} \tag{5.2}$$

where K^+ is the set of cluster combinations that contains at least one true ground atom, and K^- is the set of cluster combinations that does not contain any true ground atoms. Observe that $|K^+| + |K^-| = |\Gamma_r||\Gamma_x||\Gamma_y|$. Even though it is tractable to compute the first summation over $|K^+|$ (which is at most the number of true ground atoms), it may not be feasible to compute the second summation over $|K^-|$ for large $|\Gamma_i|$s. Hence, for tractability, we assume that all tuples in K^- belong to a single 'default' cluster combination with the same probability p_{false} of being false. The log-posterior is simplified as

$$\begin{aligned}\log P(\Gamma|R) = &\sum_{k\in K^+}\left[t_k \log\left(\frac{t_k+\alpha}{t_k+f_k+\alpha+\beta}\right) + f_k \log\left(\frac{f_k+\beta}{t_k+f_k+\alpha+\beta}\right)\right] \\ &+ \left(|S_r||S_x||S_y| - \sum_{k\in K^+}(t_k+f_k)\right)\log(p_{false}) - \lambda m^+_{cc} + \mu d + \mathcal{C}'\end{aligned} \tag{5.3}$$

where S_i is the set of symbols of type i, $(|S_r||S_x||S_y| - \sum_{k\in K^+}(t_k+f_k))$ is the number of (false) tuples in K^-, m^+_{cc} is the number of cluster combinations containing at least one true ground atom, and $\mathcal{C}' = \mathcal{C} - \lambda$.

SNE simplifies the learning problem by performing hard assignment of symbols to clusters (i.e., instead of computing probabilities of cluster membership, a symbol is simply assigned to its most likely cluster). Since, given a cluster assignment, the MAP weights can be computed in closed

form, SNE simply searches over cluster assignments, evaluating each assignment by its posterior probability.

SNE uses a bottom-up agglomerative clustering algorithm to find the MAP clustering (Table 5.1). The algorithm begins by assigning each symbol to its own unit cluster. Next we try to merge pairs of clusters of each type. We create candidate pairs of clusters, and for each of them, we evaluate the change in log posterior probability (Equation 5.3) if the pair is merged. If the candidate pair improves log posterior probability, we store it in a sorted list. We then iterate through the list, performing the best merges first and ignoring those containing clusters that have already been merged. In this manner, we incrementally merge clusters until no merges can be performed to improve log posterior probability.

To avoid creating all possible candidate pairs of clusters of each type (which is quadratic in the number of clusters), we make use of canopies [62]. A canopy for relation symbols is a set of clusters such that there exist object clusters γ_x and γ_y, and for all clusters γ_r in the canopy, the cluster combination $(\gamma_r, \gamma_x, \gamma_y)$ contains at least one true ground atom $r(x, y)$. We say that the clusters in the canopy share the *property* (γ_x, γ_y). Canopies for object symbols x and y are similarly defined. We only try to merge clusters in a canopy that is no larger than a parameter $CanopyMax$. This parameter limits the number of candidate cluster pairs we consider for merges, making our algorithm more tractable. Furthermore, by using canopies, we only try 'good' merges, because symbols in clusters that share a property are more likely to belong to the same cluster than those in clusters with no property in common.

Note that we can efficiently compute the change in log posterior probability (ΔP in Table 5.1) by only considering the cluster combinations with true ground atoms that contain the merged clusters γ and γ'. Below we give the equation for computing ΔP when we merge relation clusters γ_r and γ'_r to form γ''_r. The equations for merging object clusters are similar. Let TF_k be a shorthand for

Table 5.1: SNE algorithm.

---

function *SNE*(S_r, S_x, S_y, R)

inputs: S_r, set of relation symbols

S_x, set of object symbols that appear as first arguments

S_y, set of object symbols that appear as second arguments

R, ground $r(x, y)$ atoms formed from the symbols in S_r, S_x, and S_y

output: a semantic network, $\{(\gamma_r, \gamma_x, \gamma_y) \in \Gamma_r \times \Gamma_x \times \Gamma_y : (\gamma_r, \gamma_x, \gamma_y)$ contains at least one true ground atom$\}$

for each $i \in \{r, x, y\}$

$\Gamma_i \leftarrow unitClusters(S_i)$

$mergeOccurred \leftarrow true$

while $mergeOccurred$

$mergeOccurred \leftarrow false$

for each $i \in \{r, x, y\}$

$CandidateMerges \leftarrow \emptyset$

for each $(\gamma, \gamma') \in \Gamma_i \times \Gamma_i$

$\Delta P \leftarrow$ change in $\log P(\{\Gamma_r, \Gamma_x, \Gamma_y\}|R)$ if γ, γ' are merged

if $\Delta P > 0$, $CandidateMerges \leftarrow CandidateMerges \cup \{(\gamma, \gamma')\}$

sort $CandidateMerges$ in descending order of ΔP

$MergedClusters \leftarrow \emptyset$

for each $(\gamma, \gamma') \in CandidateMerges$

if $\gamma \notin MergedClusters$ and $\gamma' \notin MergedClusters$

$\Gamma_i \leftarrow (\Gamma_i \setminus \{\gamma, \gamma'\}) \cup \{\gamma \cup \gamma'\}$

$MergedClusters \leftarrow MergedClusters \cup \{\gamma\} \cup \{\gamma'\}$

$mergedOccurred \leftarrow true$

return $\{(\gamma_r, \gamma_x, \gamma_y) \in \Gamma_r \times \Gamma_x \times \Gamma_y : (\gamma_r, \gamma_x, \gamma_y)$ contains at least one true ground atom$\}$

$t_k \log(\frac{t_k+\alpha}{t_k+f_k+\alpha+\beta}) + f_k \log(\frac{f_k+\beta}{t_k+f_k+\alpha+\beta})$.

$$\begin{aligned}
\Delta P \;=\; & \sum_{(\gamma_r^{''},\gamma_1,\gamma_2)\in K^{+}_{\gamma_r^{''}\gamma_r^{'}\gamma_r}} \Big[TF_{(\gamma_r^{''},\gamma_1,\gamma_2)} - TF_{(\gamma_r^{'},\gamma_1,\gamma_2)} - TF_{(\gamma_r,\gamma_1,\gamma_2)} + \lambda\Big] \\
& + \sum_{(\gamma_r^{''},\gamma_1,\gamma_2)\in K^{+}_{\gamma_r^{''}\cdot\gamma_r}} \Big[TF_{(\gamma_r^{''},\gamma_1,\gamma_2)} - f_{(\gamma_r^{'},\gamma_1,\gamma_2)} \log(p_{false}) - TF_{(\gamma_r,\gamma_1,\gamma_2)}\Big] \\
& + \sum_{(\gamma_r^{''},\gamma_1,\gamma_2)\in K^{+}_{\gamma_r^{''}\gamma_r^{'}\cdot}} \Big[TF_{(\gamma_r^{''},\gamma_1,\gamma_2)} - TF_{(\gamma_r^{'},\gamma_1,\gamma_2)} - f_{(\gamma_r,\gamma_1,\gamma_2)} \log(p_{false})\Big] \\
& - \mu|\gamma_r{}'||\gamma_r| \qquad (5.4)
\end{aligned}$$

where $K^{+}_{\gamma_r^{''}\gamma_r^{'}\gamma_r}$ is the set of cluster combinations with true ground atoms such that each cluster combination $(\gamma_r^{''},\gamma_1,\gamma_2)$ in the set has the property that $(\gamma_r^{'},\gamma_1,\gamma_2)$ and $(\gamma_r,\gamma_1,\gamma_2)$ also contains true atoms. $K^{+}_{\gamma_r^{''}\cdot\gamma_r}$ is the set of cluster combinations with true ground atoms such that each cluster combination $(\gamma_r^{''},\gamma_1,\gamma_2)$ in the set has the property that $(\gamma_r,\gamma_1,\gamma_2)$, but not $(\gamma_r^{'},\gamma_1,\gamma_2)$, contains true ground atoms. $K^{+}_{\gamma_r^{''}\gamma_r^{'}\cdot}$ is similarly defined. Observe that we only sum over cluster combinations with true ground atoms that contains the affected clusters γ_r, $\gamma_r^{'}$ and $\gamma_r^{''}$, rather than over all cluster combinations with true ground atoms.

SNE and the Multiple Relational Clustering (MRC) system (described in the previous chapter) are both able to simultaneously cluster objects and relations without requiring the number of clusters to be specified in advance. However, unlike SNE, MRC is able to find multiple clusterings, rather than just one. MRC also differs from SNE in having an exponential prior on the number of clusters rather than on the number of cluster combinations with true ground atoms. The main difference between SNE and MRC is in the search algorithm used. MRC calls itself recursively to find multiple clusterings. We can view MRC as growing a tree of clusterings, and it returns the finest clusterings at the leaves. In each recursive call, MRC uses a top-down generate-and-test greedy algorithm with restarts to find the MAP clustering of the subset of relation and constant symbols it received. While this 'blind' generate-and-test approach may work well for small datasets, it will not be feasible for large Web-scale datasets like the one used in our experiments. For such large datasets, the search space will be so enormous that the top-down algorithm will generate too many candidate moves to be tractable. In our experiments, we replaced MRC's search algorithm with the algorithm in Table 5.1.

5.3 Experiments

Our goal is to create a system that is capable of extracting semantic networks from what is arguably the largest and most accessible text resource — the Web. Thus in our experiments, we use a large Web corpus to evaluate the effectiveness of SNE's relational clustering approach in extracting a simple semantic network from it. Since to date, no other system could do the same, we had to modify three other relational clustering approaches so that they could run on our large Web-scale dataset, and compared SNE to them. The three approaches are Multiple Relational Clusterings (MRC; Chapter 4), Infinite Relational Model (IRM; Section 4.1) [43] and Information-Theoretic Co-clustering (ITC) [23].

We use MRC1 to denote an MRC model that is restricted to finding a single clustering. As mentioned earlier, MRC's top-down search is not feasible for large Web-scale data, so we replace its search algorithm with the one in Table 5.1. Likewise, we replace IRM's search algorithm, which is similar to MRC's, with the algorithm in Table 5.1. We also fixed the values IRM's CRP and Beta parameters. As in SNE, we assumed that the atoms in cluster combinations with only false atoms belonged to a default cluster combination, and they had the same probability p_{false} of being false. We also experimented with a CRP prior on cluster combinations. We use IRM-C and IRM-CC to respectively denote the IRM with a CRP prior on clusters, and the IRM with a CRP prior on cluster combinations.

The ITC model clusters discrete data in a two-dimensional matrix along both dimensions simultaneously. It greedily searches for the hard clusterings that optimize the mutual information between the row and column clusters. The model has been shown to perform well on noisy and sparse data. ITC's top-down search algorithm has the flavor of K-means and requires the number of row and column clusters to be specified in advance. At every step, ITC finds the best cluster for each row or column by iterating through all clusters. This will not be tractable for large datasets like our Web dataset, which can contain many clusters. Thus, we instead use the algorithm in Table 5.1 (ΔP in Table 5.1 is set to the change in mutual information rather than the change in log posterior probability). Notice that, even if ITC's search algorithm were tractable, we would not be able to apply it to our problem because it only works on two-dimensional data. We extend ITC to three dimensions by optimizing the mutual information among the clusters of three dimensions. Furthermore,

since we do not know the exact number of clusters in our Web dataset a priori, we follow [23]'s suggestion of using an information-theoretic prior (BIC [93]) to select the appropriate number of clusters. We use ITC-C and ITC-CC to respectively denote the model with a BIC prior on clusters and the model with a BIC prior on cluster combinations. Note that, unlike SNE, ITC does not give a probability distribution over possible worlds, which we need in order to do inference and answer queries (although that is not the focus of this paper).

5.3.1 Dataset

We compared the various models on a dataset of about 2.1 million triples[1] extracted in a Web crawl by TextRunner [3]. Each triple takes the form $r(x, y)$ where r is a relation symbol, and x and y are object symbols. Some example triples are: $\texttt{named\_after}(\texttt{Jupiter}, \texttt{Roman\_god})$ and $\texttt{upheld}(\texttt{Court}, \texttt{ruling})$. There are 15,872 distinct r symbols, 700,781 distinct x symbols, and 665,378 distinct y symbols. Two characteristics of TextRunner's extractions are that they are sparse and noisy. To reduce the noise in the dataset, our search algorithm (Table 5.1) only considered symbols that appeared at least 25 times. This leaves 10,214 r symbols, 8942 x symbols, and 7995 y symbols. There are 2,065,045 triples that contain at least one symbol that appears at least 25 times. In all experiments, we set the $CanopyMax$ parameter to 50. We make the closed-world assumption for all models (i.e., all triples not in the dataset are assumed false).

5.3.2 SNE vs. MRC

We compared the performances of SNE and MRC1 in learning a *single* clustering of symbols. We set the λ, μ and p_{false} parameters in SNE to 100, 100 and 0.9999 respectively based on preliminary experiments. We set SNE's α and β parameters to 2.81×10^{-9} and $10 - \alpha$ so that $\frac{\alpha}{\alpha+\beta}$ is equal to the fraction of true triples in the dataset. (A priori, we should predict the probability that a ground atom is true to be this value.) We evaluated the clusterings learned by each model against a gold standard manually created by the first author. The gold standard assigns 2688 r symbols, 2568 x symbols and 3058 y symbols to 874, 511 and 700 non-unit clusters respectively. We measured the pairwise precision, recall and F1 of each model against the gold standard. Pairwise precision is the

[1] Publicly available at http://knight.cis.temple.edu/~yates/data/resolver_data.tar.gz

fraction of symbol pairs in learned clusters that appear in the same gold clusters. Pairwise recall is the fraction of symbol pairs in gold clusters that appear in the same learned clusters. F1 is the harmonic mean of precision and recall. For the weight of MRC1's exponential prior on clusters, we tried the following values and pick the best: 0, 1, 10–100 (in increments of 10), and 110–1000 (in increments of 100). We report the precision, recall and F1 scores that are obtained with the best value of 80. From Table 5.2, we see that SNE performs significantly better than MRC1.

We also ran MRC to find multiple clusterings. Since the gold standard only defines a single clustering, we cannot use it to evaluate the multiple clusterings. We provide a qualitative evaluation instead. MRC returns 23,151 leaves that contain non-unit clusters, and 99.8% of these only contain 3 or fewer clusters of size 2. In contrast, SNE finds many clusters of varying sizes (see Table 5.6). The poor performance of MRC in finding multiple clusterings is due to data sparsity. In each recursive call to MRC, it only receives a small subset of the relation and object symbols. Thus with each call the data becomes sparser, and there is not enough signal to cluster the symbols.

5.3.3 Joint vs. Separate Clustering of Relations and Objects

We investigated the effect of having SNE only cluster relation symbols, first-argument object symbols, or second-argument object symbols, e.g., if SNE cluster relation symbols, then it does not cluster both kinds of object symbols. From Table 5.3, we see that SNE obtains a significantly higher F1 when it clusters relations and objects jointly than when it clusters them separately.

5.3.4 SNE vs. IRM and ITC

We compared IRM-C and IRM-CC with respect to the gold standard. We set IRM's Beta parameters to the values of SNE's α and β, and set p_{false} to the same value as SNE's. We tried the following values for the parameter of the CRP priors: 0.25, 0.5, 0.75, 1–10 (in increments of 1), 20–100 (in increments of 10). We found that the graphs showing how precision, recall, and F1 vary with the CRP value are essentially flat for both IRM-C and IRM-CC. Both system perform about the same. The slightly higher precision, recall, and F1 scores occur at the low end of the values we tried, and we use the best one of 0.25 for the slightly better-performing IRM-CC system. Henceforth, we denote this IRM as IRM-CC-0.25 and use it for other comparisons.

Table 5.2: Comparison of SNE and MRC1 performances on gold standard. Object 1 and Object 2 respectively refer to the object symbols that appear as the first and second arguments of relations. The best F1s are shown in bold.

	Relation			**Object 1**			**Object 2**		
Model	Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall	F1
SNE	0.452	0.187	**0.265**	0.460	0.061	**0.108**	0.558	0.062	**0.112**
MRC1	0.054	0.044	0.049	0.031	0.007	0.012	0.059	0.011	0.018

Table 5.3: Comparison of SNE performance when it clusters relation and object symbols jointly and separately. SNE-Sep clusters relation and object symbols separately. Object 1 and Object 2 respectively refer to the object symbols that appear as the first and second arguments of relations. The best F1s are shown in bold.

	Relation			**Object 1**			**Object 2**		
Model	Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall	F1
SNE	0.452	0.187	**0.265**	0.460	0.061	**0.108**	0.558	0.062	**0.112**
SNE-Sep	0.597	0.116	0.194	0.519	0.045	0.083	0.551	0.047	0.086

We also compared SNE, IRM-CC-0.25, ITC-C and ITC-CC. From Table 5.4, we see that ITC performs better with a BIC prior on cluster combinations than a BIC prior on clusters. We also see that SNE performs the best in terms of F1.

We then evaluated SNE, IRM-CC-0.25 and ITC-CC in terms of the semantic statements that they learned. A cluster combination that contains a true ground atom corresponds to a semantic statement. SNE, IRM-CC-0.25 and ITC-CC respectively learned 1,464,965, 1,254,995 and 82,609 semantic statements. We manually inspected semantic statements containing 5 or more true ground atoms and counted the number that were correct. Table 5.5 shows the results. Even though SNE's accuracy is smaller than IRM-CC-0.25's and ITC-CC's by 11% and 7% respectively, SNE more than compensates for the lower accuracy by learning 127% and 273% more correct statements respectively. Figure 5.1 shows examples of correct semantic statements learned by SNE.

SNE, IRM-CC-0.25 and ITC-CC respectively ran for about 5.5 hours, 9.5 hours, and 3 days on identically configured machines. ITC-CC spent most of its time computing the mutual information among three clusters. To compute the mutual information, given any two clusters, we have to retrieve the number of cluster combinations that contain the two clusters. Because of the large number of cluster pairs, we choose to use a data structure (red-black tree) that is space-efficient, but pays a time penalty when looking up the required values.

5.3.5 Comparison of SNE with WordNet

We also compared the object clusters that SNE learned with WordNet [32], a hand-built semantic lexicon for the English language. WordNet organizes 117,798 distinct nouns into a taxonomy of 82,115 concepts. There are respectively 4883 first-argument and 5076 second-argument object symbols that appear at least 25 times in our dataset and also in WordNet. We converted each node (synset) in WordNet's taxonomy into a cluster containing its original concepts and all its children concepts. We then matched each SNE cluster to the WordNet cluster that gave the best F1 score. We measured F1 as the harmonic mean of precision and recall. Precision is the fraction of symbols in an SNE cluster that is also in the matched WordNet cluster. Recall is the fraction of symbols in a WordNet cluster that is also in the corresponding SNE cluster. Table 5.6 shows how precision, recall and F1 vary with cluster sizes. (The scores are averaged over all object clusters of the same

Table 5.4: Comparison of SNE, IRM-CC-0.25, ITC-CC and ITC-C performances on gold standard. Object 1 and Object 2 respectively refer to the object symbols that appear as the first and second arguments of relations. The best F1s are shown in bold.

	Relation			**Object 1**			**Object 2**		
Model	Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall	F1
SNE	0.452	0.187	**0.265**	0.461	0.061	**0.108**	0.558	0.062	**0.112**
IRM-CC-0.25	0.201	0.089	0.124	0.252	0.043	0.073	0.307	0.041	0.072
ITC-CC	0.773	0.003	0.006	0.470	0.047	0.085	0.764	0.002	0.004
ITC-C	0.000	0.000	0.000	0.571	0.000	0.000	0.333	0.000	0.000

Table 5.5: Evaluation of semantic statements learned by SNE, IRM-CC-0.25, and ITC-CC.

Model	**Total Statements**	**Num. Correct**	**Fract. Correct**
SNE	1241	965	0.778
IRM-CC-0.25	487	426	0.874
ITC-CC	310	259	0.835

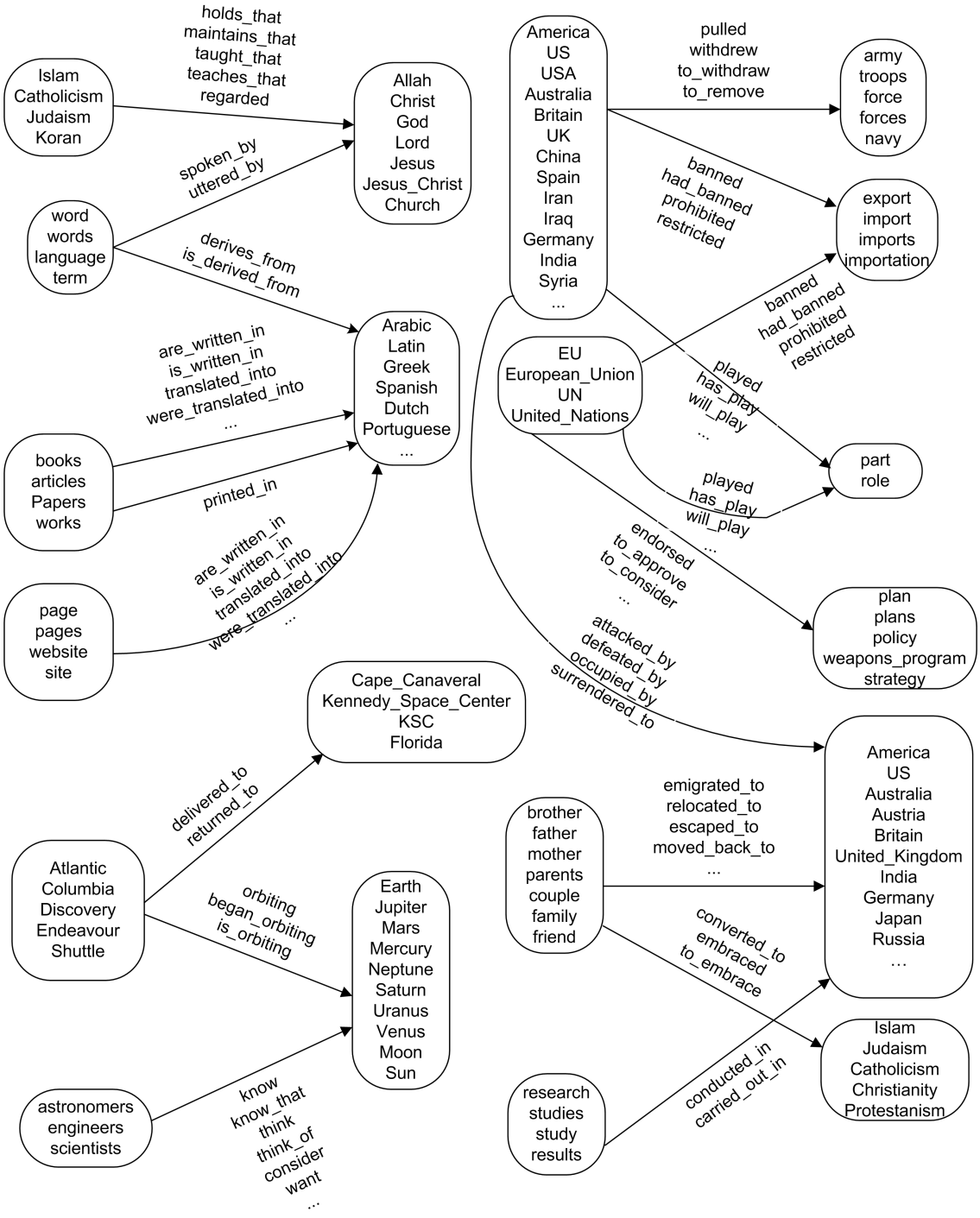

Figure 5.1: Fragments of a semantic network learned by SNE. Nodes are concept clusters, and the labels of links are relation clusters.

size). We see that the F1s are fairly good for object clusters of size 7 or less. The table also shows how the level of the matched cluster in WordNet's taxonomy vary with cluster size. The higher the level, the more specifc the concept represented by the matched WordNet cluster. For example, clusters at level 7 correspond to specific concepts like 'country', 'state', 'dwelling', and 'home', while the single cluster at level 0 (i.e., at the root of the taxonomy) corresponds to 'all entities'. We see that the object clusters correspond to fairly specifc concepts in WordNet. We did not compare the relation clusters to WordNet's verbs because the overlap between the relation symbols and the verbs are too small.

5.4 Related Work

Rajaraman and Tan [81] propose a system that learns a semantic network by clustering objects but not relations. While it anecdotally shows a snippet of its semantic network, an empirical evaluation of the network is not reported. Hasegawa et al. [39] propose an unsupervised approach to discover relations from text. They treat the short text segment between each pair of objects as a relation and cluster pairs of objects using the similarity between their relation strings. Each cluster corresponds to a relation, and a pair of objects can appear in at most one cluster (relation). In contrast, SNE allows a pair of objects to participate in multiple relations (semantic statements). Shinyama and Sekine [94] form (possibly overlapping) clusters of tuples of objects (rather than just pairs of objects). They use the words surrounding the objects in the same sentence to form a pattern. Objects in sentences with the same pattern are deemed to be related in the same way and are clustered together. All three previous systems are not domain-independent because they rely on name entity (NE) taggers to identify objects in text. The concepts and relations that they learn are restricted by the object types that can be identified with the NE taggers. All three systems also use ad-hoc techniques that do not give a probability distribution over possible worlds, which we need in order to perform inference and answer queries. By only forming clusters of (tuples of) objects and not relations, they do not explicitly learn high-level relations like SNE.

ALICE [4] is a system for lifelong knowledge extraction from a Web corpus. Like SNE, it uses TextRunner's triples as input. However, unlike SNE, it requires background knowledge in the form of an existing domain-specific concept taxonomy and does not cluster relations into higher level

Table 5.6: Comparison of SNE object clusters with WordNet.

Cluster Size	**Num. Clusters**	**Level**	**Prec.**	**Recall**	**F1**
47	1	7.0±0.0	0.8±0.0	0.2±0.0	0.4±0.0
36	1	8.0±0.0	0.3±0.0	0.3±0.0	0.3±0.0
24	1	6.0±0.0	0.2±0.0	0.3±0.0	0.2±0.0
19	1	7.0±0.0	0.2±0.0	0.3±0.0	0.2±0.0
16	1	7.0±0.0	0.3±0.0	0.3±0.0	0.3±0.0
12	3	7.0±0.7	0.5±0.1	0.7±0.1	0.5±0.2
11	1	6.0±0.0	0.9±0.0	0.7±0.0	0.8±0.0
10	2	5.5±0.7	0.6±0.1	0.9±0.1	0.5±0.1
8	5	7.0±0.9	0.4±0.2	0.7±0.4	0.3±0.1
7	4	6.0±1.4	0.7±0.3	0.8±0.2	0.9±0.1
6	12	6.6±1.7	0.4±0.2	0.6±0.2	0.6±0.2
5	12	7.2±1.6	0.4±0.2	0.5±0.3	0.7±0.1
4	84	7.2±1.7	0.4±0.1	0.7±0.2	0.6±0.2
3	185	7.3±1.8	0.5±0.2	0.7±0.2	0.7±0.2
2	1419	7.2±1.8	0.6±0.1	0.7±0.1	0.8±0.1

ones.

RESOLVER [107] is a system that takes TextRunner's triples as input and resolves references to the same object and relations by clustering the references together (e.g., `Red_Planet` and `Mars` are clustered together). In contrast, SNE learns abstract concepts and relations (e.g., `Mars`, `Venus`, `Earth`, etc. are clustered together to form the concept of 'planet'). Further, SNE learns a semantic network but RESOLVER does not. Unlike SNE, RESOLVER's probabilistic model clusters objects and relations separately rather than jointly. To allow information to propagate between object clusters and relation clusters, RESOLVER uses an ad-hoc approach. In its experiments, RESOLVER gives similar results with or without the ad-hoc approach. In contrast, we show in our experiments that SNE gives better performance with joint rather than separate clustering (see Table 5.3). In a preliminary experiment where we adapt SNE to only use string similarities between objects (and relations), we find that SNE performs better than RESOLVER on an entity resolution task on the dataset described in Section 5.3.

LDA-SP [87] is a system that uses the LinkLDA model [30] to cluster the arguments of TextRunner's $r(x, y)$ triples. LinkLDA defines a generative model for the triples by endowing each relation r with a T-dimensional multinomial θ_r over topics. Each θ_r is drawn from a Dirichlet with parameter α. For each triple, two topics c_x and c_y are sampled according to θ_r, and then the observed arguments x and y are chosen according to multinomials β_x and β_y. In LinkLDA, a topic can be interpreted as a (soft) cluster of arguments. (LinkLDA is an extension of the LDA model [10], which draws one topic from θ_r rather than two.)

The differences between LDA-SP and SNE can be seen by comparing their joint probabilites.

In LDA-SP, the joint probability is given by

$$P(R, \Theta, C|\alpha, \beta_x, \beta_y) = \prod_{r=1}^{M} p(\theta_r) \prod_{i=1}^{N_r} p(c^i_{r,x}|\theta_r)p(c^i_{r,y}|\theta_r)p(x^i_r|c^i_{r,x}, \beta_x)p(y^i_r|c^i_{r,y}, \beta_y) \tag{5.5}$$

where R is a set of triples, $\Theta = \{\theta_r\}$, $C = \{(c^i_x, c^i_y)\}$, M is the number of relations, N_r is the number of triples with r as their relation, $c^i_{r,x}$ and $c^i_{r,y}$ are respectively the topics chosen for the x and y arguments of the ith triple of relation r, and x^i_r and y^i_r are respectively the x and y arguments of the ith triple of relation r. The prior term is $\prod_{r=1}^{M} p(\theta_r) \prod_{i=1}^{N_r} p(c^i_{r,x}|\theta_r)p(c^i_{r,y}|\theta_r)$ and the likelihood term is $\prod_{r=1}^{M} \prod_{i=1}^{N_r} p(x^i_r|c^i_{r,x}, \beta_x)p(y^i_r|c^i_{r,y}, \beta_y)$.

In SNE, the joint probability is given by

$$P(R,\Gamma) = \left(\prod_{k\in K}\left[\left(\frac{t_k}{t_k+f_k}\right)^{t_k}\left(\frac{f_k}{t_k+f_k}\right)^{f_k}\right]\right)\exp(-\lambda m_{cc}+\mu d) \tag{5.6}$$

where R is a set of triples, Γ is the assignment of relation and argument symbols to clusters, K is the set of cluster combinations, t_k and f_k are respectively the number of true and false triples in cluster combination k, λ and μ are user-specified weights of the rules in the prior MLN (see Section 5.2), m_{cc} is the number of cluster combinations, and d is the number of pairs of symbols that belong to different clusters. The prior term is $\exp(-\lambda m_{cc}+\mu d)$ and the likelihood term is $\prod_{k\in K}\left[\left(\frac{t_k}{t_k+f_k}\right)^{t_k}\left(\frac{f_k}{t_k+f_k}\right)^{f_k}\right]$.

First, we compare the prior terms of the two joint likelihoods. In LDA-SP, the topics (clusters) have a multinomial prior, whereas in SNE, the clusters have a simple exponential prior. In LDA-SP, the topics for the two arguments are assumed to be drawn from the same T-dimensional multinomial. Consequently, both arguments have the same number T of topics. Such a modelling assumption may be appropriate for data extracted from the Web (e.g., TextRunner's triples) because they frequently contain a relation together with its inverse (e.g., $\texttt{BornIn}(\texttt{Peter},\texttt{USA})$ and $\texttt{LocationOfBirth}(\texttt{USA},\texttt{Peter})$). Hence the topics for the two arguments are likely to be symmetric. However, for data that do not generally contain both relations and their inverses, SNE could potentially perform better because it allows the arguments to have different numbers of clusters to better model the data.

Next, we compare the likelihood terms. In LDA-SP, each *argument* is drawn from a *multinomial*. From the form of SNE's likelihood term, we can see that each *triple* $r(x,y)$ is drawn from a *binomial* (which corresponds to a cluster combination).

The choice of whether LDA-SP's multinomial-likelihood-cum-Dirichlet-prior model or SNE's binomial-likelihood-cum-exponential-prior model is the better representation for a dataset can be determined by their relative empirical performances on held-out data. Also note that SNE is able to cluster relation symbols but LDA-SP cannot.

SNE and LDA-SP also differ in how they learn their model parameters and cluster assignments. SNE uses *MAP estimation* to learn the most likely parameters and assignment of symbols to clusters (thus a symbol can belong to exactly one cluster). By using MAP estimation, SNE's joint proba-

bility (and posterior probability) can be computed efficiently in closed-form, thus permitting SNE to tractably search for the MAP cluster assignment. In contrast, LDA-SP addresses the task of *full Bayesian learning* by using MCMC methods to find a distribution over parameters (including a distribution of cluster membership). However, MCMC methods are computationally expensive and are unlikely to scale well to large datasets. In our empirical experiments, SNE took only 5.5 hours on a dataset with 2.1 million triples. However, LDA-SP took 11 days on another dataset of comparable size[2].

Hierarchical LDA (hLDA) [9, 83] is an extension of LDA that is able to learn a hierarchy of topics. hLDA directly models a topic hierarchy by including it as a variable in its joint likelihood. In the joint likelihood, hLDA uses a variant of LDA for its likelihood component, and the nested Chinese Restaurant Process as a prior on the structure of the topic hierarchy. In contrast, SNE and MRC do not model a hierarchy of clusters. (Even though MRC's search process creates a hierarchy of cluster combinations, MRC does not define a probability model of the hierarchy.) Extending SNE and MRC to model a cluster hierarchy is an item of future work.

5.5 Conclusion

In this chapter, we presented SNE, a scalable, unsupervised, domain-independent system for extracting knowledge in the form of simple semantic networks from text. SNE is based on second-order Markov logic. It uses a bottom-up agglomerative clustering algorithm to jointly cluster relation symbols and object symbols into high-level relations and concepts, and allows information to propagate between the clusters as they are formed. Empirical comparisons with three systems on a large real-world Web dataset show the promise of our approach.

In the next chapter, we adapt the ideas developed in SNE to the problem of learning good MLN formulas.

[2]Personal communication with Alan Ritter.

Chapter 6

LEARNING MARKOV LOGIC NETWORK STRUCTURE VIA HYPERGRAPH LIFTING

6.1 Introduction

In Chapter 3, we proposed an approach for learning MLN structure that systematically enumerates candidate clauses by starting from an empty clause, greedily adding literals to it, and testing the resulting clause's empirical fit to training data. Such a strategy has two shortcomings: searching the large space of clauses is computationally expensive; and it is susceptible to converging to a local optimum, missing potentially useful clauses. These shortcomings can be ameliorated by using the data to *a priori* constrain the space of candidates. This is the basic idea in *relational pathfinding* [84], which finds paths of true ground atoms that are linked via their arguments and then generalizes them into first-order rules. Each path corresponds to a conjunction that is true at least once in the data. Since most conjunctions are false, this helps to concentrate the search on regions with promising rules. However, pathfinding potentially amounts to exhaustive search over an exponential number of paths. Hence, systems using relational pathfinding (e.g., BUSL [66]) typically restrict themselves to very short paths, creating short clauses from them and greedily joining them into longer ones.

In this chapter, we present LHL, an approach that uses relational pathfinding to a fuller extent than previous ones. It mitigates the exponential search problem by first inducing a more compact representation of data, in the form of a hypergraph over clusters of constants, using the techniques introduced in Chapters 4 and 5. Pathfinding on this 'lifted' hypergraph is typically at least an order of magnitude faster than on the ground training data and produces MLNs that are more accurate than previous state-of-the-art approaches. LHL is short for Learning via Hypergraph Lifting.

In the next section, we present LHL in more detail. After that we report our experiments (Section 6.3) and discuss related work (Section 6.4).

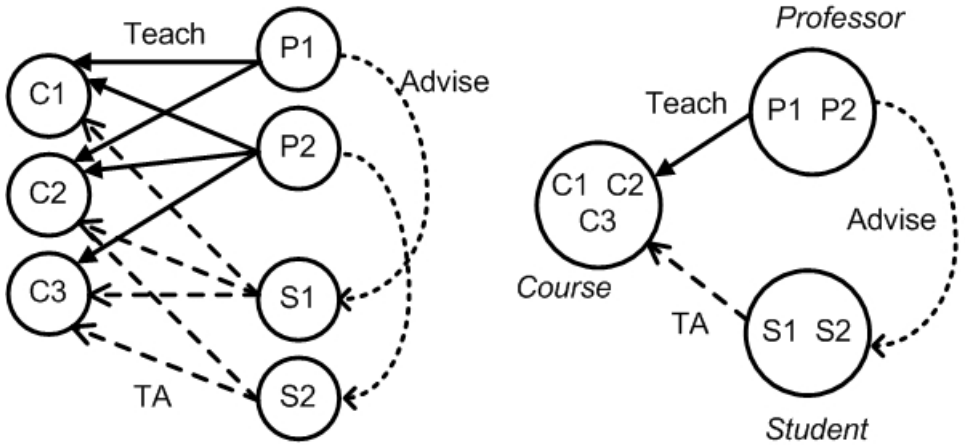

Figure 6.1: Lifting a hypergraph.

6.2 Learning via Hypergraph Lifting

In LHL, we make use of *hypergraphs*. A hypergraph is a straightforward generalization of a graph in which an edge can link any number of nodes, rather than just two. More formally, we define a hypergraph as a pair (V, E) where V is a set of nodes, and E is a set of labeled non-empty subsets of V called hyperedges. In LHL, we find *paths* in a hypergraph. A path is defined as a set of hyperedges such that for any two hyperedges e_0 and e_n in the set, there exists an ordering of (a subset of) hyperedges in the set $e_0, e_1, \ldots, e_{n-1}, e_n$ such that e_i and e_{i+1} share at least one node.

A database can be viewed as a hypergraph with constants as nodes and true ground atoms as hyperedges. Each hyperedge is labeled with a predicate symbol. Nodes (constants) are linked by a hyperedge (true ground atom) if and only if they appear as arguments in the hyperedge. (Henceforth we use *node* and *constant* interchangeably, and likewise for *hyperedge* and *true ground atom*.) A path of hyperedges can be generalized into a first-order clause by variabilizing their arguments. To avoid tracing the exponential number of paths in the hypergraph, LHL first jointly clusters the nodes into higher-level concepts, and by doing so it also clusters the hyperedges (i.e., the ground atoms containing the clustered nodes). The 'lifted' hypergraph has fewer nodes and hyperedges, and therefore fewer paths, reducing the cost of finding them.

Figure 6.1 provides an example. We have a database describing an academic department where

professors tend to have students whom they are advising as teaching assistants (TAs) in the classes the professors are teaching. The left graph is created from the database, and after lifting, results in the right graph. Observe that the lifted graph is simpler and the clustered constants correspond to the high-level concepts of *Professor*, *Student* and *Course*.

Table 6.1 gives the pseudocode for LHL. LHL begins by lifting a hypergraph (Table 6.2). Then it finds paths in the lifted hypergraph (Table 6.3). Finally it creates candidate clauses from the paths and learns their weights to create an MLN (Table 6.4). We describe each component of LHL in turn.

6.2.1 Hypergraph Lifting

We call our hypergraph lifting algorithm LiftGraph. LiftGraph is similar to the MRC and SNE algorithms [46, 47] (described in Chapter 4 and 5 respectively). It differs from them in the following ways. LiftGraph can handle relations of arbitrary arity, whereas SNE can only handle binary relations. Unlike MRC, LiftGraph finds a single clustering of constant symbols rather than multiple clusterings. While both SNE and MRC can cluster predicate symbols, in this paper, for simplicity, we do not cluster predicates. (However, it is straightforward to extend LiftGraph to do so.) Most domains contain many fewer predicates than objects, and structure learning alone suffices to capture the dependencies among them, which is what LHL does. (Because SNE and MRC do not have a structure learning component, it is essential for them to cluster predicates in order to learn the dependencies among them.)

LiftGraph works by jointly clustering the constants in a hypergraph in a bottom-up agglomerative manner, allowing information to propagate from one cluster to another as they are formed. The number of clusters need not be pre-specified. As a consequence of clustering the constants, the ground atoms in which the constants appear are also clustered. In Figure 6.1, the hyperedge `Teach`(*Professor,Course*) in the lifted hypergraph contains the ground atoms `Teach(P1, C1)`, `Teach(P1, C2)`, `Teach(P2, C1)`, etc. Each hyperedge in the lifted hypergraph contains at least one true ground atom.

LiftGraph is defined using Markov logic. We use the variable r to represent a predicate, x_i for the ith argument of a predicate, γ_i for a cluster of ith arguments of a predicate (i.e., a set of constant symbols), and Γ_t for a clustering of constant symbols of type t (i.e., a set of clusters

Table 6.1: LHL algorithm.

---

function *LHL*$(D, T, \omega, \mu, \nu, \pi, \theta_{atoms})$

input: D, a relational database

T, a set of types, where a type is a set of constants

ω, maximum number of hyperedges in a path

μ, minimum number of ground atoms per hyperedge in a path in order for it to be selected

ν, maximum number of ground atoms to sample in a path

π, length penalty on clauses

θ_{atoms}, fraction of atoms to sample from D

output: $(Clauses, Weights)$, an MLN containing a set of learned clauses and their weights

note: Index H maps from each node γ_i to the set of hyperedges $r(\gamma_1, \ldots, \gamma_i, \ldots, \gamma_n)$ containing γ_i

E is a set of hyperedges in a lifted hypergraph

$Paths$ is a set of paths, each path being a set of hyperedges

$(H, E) \leftarrow LiftGraph(D, T)$

$Paths \leftarrow \emptyset$

for each $r(\gamma_1, \ldots, \gamma_n) \in E$

$Paths \leftarrow Paths \cup FindPaths(\{r(\gamma_1, \ldots, \gamma_n)\}, \{\gamma_1, \ldots, \gamma_n\}, \omega, H)$

$(Clauses, Weights) \leftarrow CreateMLN(Paths, D, \mu, \nu, \pi, \theta_{atoms})$

return $(Clauses, Weights)$

Table 6.2: LiftGraph algorithm.

function *LiftGraph*(D, T)

  note: The inputs and output are as described in Table 6.1

for each $t \in T$

  $\Gamma_t \leftarrow \emptyset$

  for each $x \in t$

    $\Gamma_t \leftarrow \Gamma_t \cup \{\gamma_x\}$ (γ_x is a unit cluster containing x)

    $H[\gamma_x] \leftarrow \emptyset$ (H maps from nodes to hyperedges)

$E \leftarrow \emptyset$ (E contains hyperedges)

for each true ground atom $r(x_1, \ldots, x_n) \in D$

  $E \leftarrow E \cup \{r(\gamma_{x_1}, \ldots, \gamma_{x_n})\}$

  for each $x_i \in \{x_1, \ldots, x_n\}$

    $H[\gamma_{x_i}] \leftarrow H[\gamma_{x_i}] \cup \{r(\gamma_{x_1}, \ldots, \gamma_{x_n})\}$

repeat

  for each $t \in T$

    $(\gamma_{best}, \gamma'_{best}) \leftarrow ClusterPairWithBestGain(\Gamma_t)$

    if $\{(\gamma_{best}, \gamma'_{best})\} \neq \emptyset$

      $\gamma_{new} \leftarrow \gamma_{best} \cup \gamma'_{best}$

      $\Gamma_t \leftarrow (\Gamma_t \setminus \{\gamma_{best}, \gamma'_{best}\}) \cup \gamma_{new}$

      $H[\gamma_{new}] \leftarrow \emptyset$

      for each $\gamma \in \{\gamma_{best}, \gamma'_{best}\}$

        for each $r(\gamma_1, \ldots, \gamma, \ldots, \gamma_n) \in H[\gamma]$

          $H[\gamma_{new}] \leftarrow H[\gamma_{new}] \cup \{r(\gamma_1, \ldots, \gamma_{new}, \ldots, \gamma_n)\}$

          $E \leftarrow E \setminus \{r(\gamma_1, \ldots, \gamma, \ldots, \gamma_n)\}$

          $E \leftarrow E \cup \{r(\gamma_1, \ldots, \gamma_{new}, \ldots, \gamma_n)\}$

        $H[\gamma] \leftarrow \emptyset$

until no clusters are merged for all t

return (H, E)

Table 6.3: FindPaths algorithm.

---

function *FindPaths*$(CurPath, V, \omega, H)$

  input: $CurPath$, set of connected hyperedges

  V, set of nodes in $CurPath$

  note: The other inputs & output are as described in Table 6.1

if $|CurPath| = \omega$

  return $\emptyset$

$Paths \leftarrow \emptyset$

for each $\gamma_i \in V$

  for each $r(\gamma_1, \ldots, \gamma_n) \in H[\gamma_i]$

    if $r(\gamma_1, \ldots, \gamma_n) \notin CurPath$

      $CurPath \leftarrow CurPath \cup \{r(\gamma_1, \ldots, \gamma_n)\}$

      $Paths \leftarrow Paths \cup \{CurPath\}$

      $V' \leftarrow \emptyset$

      for each $\gamma_j \in \{\gamma_1, \ldots, \gamma_n\}$

        if $\gamma_j \notin V$

          $V \leftarrow V \cup \{\gamma_j\}$

          $V' \leftarrow V' \cup \{\gamma_j\}$

      $Paths \leftarrow Paths \cup FindPath(CurPath, V, \omega, H)$

      $CurPath \leftarrow CurPath \setminus \{r(\gamma_1, \ldots, \gamma_n)\}$

      $V \leftarrow V \setminus V'$

return $Paths$

Table 6.4: CreateMLN algorithm.

function *CreateMLN*$(Paths, D, \mu, \nu, \pi, \theta_{atoms})$

calls: $VariabilizePaths(Paths)$, replaces the nodes in each path in $Paths$ with variables

$MakeClauses(Path)$, creates clauses from hyperedges in $Path$

$Sample(Path, \nu)$, uniformly samples ν ground atoms from $Path$

$SampleDB(D, \theta_{atoms})$, uniformly samples a fraction θ_{atoms} of atoms from database D

$NumTrueGroundAtoms(Path)$, counts the number of true ground atoms in $Path$

note: The inputs and output are as described in Table 6.1

(only select paths with enough true ground atoms (heuristic 1))

$Paths \leftarrow VariabilizePaths(Paths)$

$SelectedPaths \leftarrow \emptyset$

for each $p \in Paths$

if $(NumTrueGroundAtoms(p) >= PathLength(p) * \mu)$

$SelectedPaths \leftarrow SelectedPaths \cup \{p\}$

(evaluate candidates with ground atoms in $Path$ (heuristic 2))

$CandidateClauses \leftarrow \emptyset$

for each $p \in SelectedPaths$

$D' \leftarrow Sample(p, \nu)$

for each $c \in MakeClauses(p)$

if $Score(c, D') > Score(\emptyset, D')$

$CandidateClauses \leftarrow CandidateClauses \cup \{c\}$

$CandidateClauses \leftarrow SortByLength(CandidateClauses)$

(Evaluate candidates with ground atoms in database D)

$D' \leftarrow SampleDB(D, \theta_{atoms})$

$SelectedClauses \leftarrow \emptyset$

for each $c \in CandidateClauses$

$BetterThanSubClauses \leftarrow True$

for each $c' \in (SubClauses(c) \cap SelectedClauses)$

if $Score(c, D') < Score(c', D')$

$BetterThanSubClauses \leftarrow False$

break

if $(BetterThanSubClauses)$

$SelectedClauses \leftarrow SelectedClauses \cup \{c\}$

$AddClausesToMLN(SelectedClauses)$

$Weights \leftarrow LearnWeights(SelectedClauses)$

return $(SelectedClauses, Weights)$

or, equivalently, a partitioning of a set of symbols). If x_i is in γ_i, we say that $(x_1, \ldots, x_n)$ is in the *cluster combination* $(\gamma_1, \ldots, \gamma_n)$, and that $(\gamma_1, \ldots, \gamma_n)$ *contains* the atom $r(x_1, \ldots, x_n)$. $r(\gamma_1, \ldots, \gamma_n)$ denotes a hyperedge connecting nodes $\gamma_1, \ldots, \gamma_n$. A hypergraph representing the true ground atoms $r(x_1, \ldots, x_n)$ in a database is simply $(V = \{\{x_i\}\}, E = \{\, r(\{x_1\}, \ldots, \{x_n\})\,\})$ with each constant x_i in its own cluster, and a hyperedge for each true ground atom.

The learning problem in LiftGraph consists of finding the cluster assignment $\{\Gamma\}$ that maximizes the posterior probability $P(\{\Gamma\}|D) \propto P(\{\Gamma\})P(D|\{\Gamma\})$, where D is a database of truth assignments to the observable $r(x_1, \ldots, x_n)$ ground atoms. The prior $P(\{\Gamma\})$ is simply an MLN containing two rules. The first rule states that each symbol belongs to exactly one cluster. This rule is hard, i.e., it has infinite weight and cannot be violated.

$$\forall x \; \exists^1 \gamma \;\; x \in \gamma$$

The second rule is

$$\forall \gamma_1, \ldots, \gamma_n \; \exists x_1, \ldots, x_n \quad x_1 \in \gamma_1 \wedge \ldots \wedge x_n \in \gamma_n$$

with negative weight $-\infty < -\lambda < 0$, which imposes an exponential prior on the number of cluster combinations to prevent overfitting. The parameter λ is fixed during learning, and is the penalty in log-posterior incurred by adding a cluster combination.

The MLN for the likelihood $P(D|\{\Gamma\})$ contains the following rules. For each predicate r and each cluster combination $(\gamma_1, \ldots, \gamma_n)$ that contains a true ground atom of r, the MLN contains the rule:

$$\forall x_1, \ldots, x_n \quad x_1 \in \gamma_1 \wedge \ldots \wedge x_n \in \gamma_n \Rightarrow r(x_1, \ldots, x_n)$$

We call these *atom prediction* rules because they state that the truth value of an atom is determined by the cluster combination it belongs to. These rules are soft. At most there can be one such rule for each true ground atom (i.e., when each constant is in its own cluster).

For each predicate r, we create a rule

$$\forall x_1, \ldots, x_n \quad \left(\textstyle\bigwedge_{i=1}^{m} \neg(x_1 \in \gamma_1^i \wedge \ldots \wedge x_n \in \gamma_n^i)\right) \Rightarrow r(x_1, \ldots, x_n)$$

where $(\gamma_1^1, \ldots, \gamma_n^1), \ldots, (\gamma_1^m, \ldots, \gamma_n^m)$ are cluster combinations containing true ground atoms of r. This rule accounts for all atoms (all false) that are not in any cluster combination with true ground

atoms of r. We call such a rule a *default* atom prediction rule because its antecedent is analogous to a default cluster combination that contains all atoms that are not in the cluster combinations of any atom prediction rule.

LiftGraph simplifies the learning problem by performing hard assignments of constant symbols to clusters (i.e., instead of computing probabilities of cluster membership, a symbol is simply assigned to its most likely cluster). The weights and the log-posterior can now be computed in closed form. The derivation of the log-posterior is given in Appendix D. LiftGraph thus simply searches over cluster assignments, evaluating each one by its posterior probability. It begins by assigning each constant symbol x_i to its own cluster $\{x_i\}$ and creating a hyperedge $r(\{x_1\}, \ldots, \{x_n\})$ for each true ground atom $r(x_1, \ldots, x_n)$. Next it creates candidate pairs of clusters of each type, and for each pair, it evaluates the gain in posterior probability if its clusters are merged. It then chooses the pair that gives the largest gain to be merged. When clusters γ_i and γ_i' are merged to form γ_i^{new}, each hyperedge $r(\gamma_1, \ldots, \gamma_i, \ldots, \gamma_n)$ is replaced with $r(\gamma_1, \ldots, \gamma_i^{new}, \ldots, \gamma_n)$ (and similarly for hyperedges containing γ_i'). Since $r(\gamma_1, \ldots, \gamma_i, \ldots, \gamma_n)$ contains at least one true ground atom, $r(\gamma_1, \ldots, \gamma_i^{new}, \ldots, \gamma_n)$ must do too. To avoid trying all possible candidate pairs of clusters, LiftGraph only tries to merge γ_i and γ_i' if they appear in hyperedges $r(\gamma_1, \ldots, \gamma_i, \ldots, \gamma_n)$ and $r(\gamma_1, \ldots, \gamma_i', \ldots, \gamma_n)$. In this manner, it incrementally merges clusters until no merges can be performed to improve posterior probability. It then returns a lifted hypergraph whose hyperedges all contain at least one true ground atom.

6.2.2 Path Finding

FindPaths constructs paths by starting from each hyperedge in a hypergraph. It begins by adding a hyperedge to an empty path, and then recursively adds hyperedges linked to nodes already present in the path (hyperedges already in the path are not re-added). Its search terminates when the path reaches a maximum length or when no new hyperedge can be added. Each time a hyperedge is added to the path, FindPaths stores the resulting path as a new one. All the paths are passed on to the next step to create clauses.

Any search algorithm can be used to find paths in the hypergraph (depth-first search, breadth-first search, iterative deepening, etc.) Our search algorithm FindPaths (Table 6.3) has the flavor of

depth-first search because it is memory efficient.

6.2.3 Clause Creation and Pruning

A path in the hypergraph corresponds to a conjunction of $r(\gamma_1, \ldots, \gamma_n)$ hyperedges. We replace each γ_i in a path with a variable, thereby creating a variabilized atom for each hyperedge. We convert the conjunction of positive literals to a clause because that is the form that is typically used by ILP (inductive logic programming) and MLN structure learning and inference algorithms. (In Markov logic, a conjunction of positive literals with weight w is equivalent to a clause of negative literals with weight $-w$). In addition, we add clauses with the signs of up to n literals flipped (where n is a user-defined parameter), since the resulting clauses may also be useful.

We evaluate each clause using weighted pseudo-log-likelihood (WPLL) given in Equation 3.1. Following the MSL experiments in Section 3.6, we weight all first-order predicates equally, penalize the WPLL with a length penalty $-\pi l$ (l is the number of literals in a clause), and penalize each clause weight with a zero-mean Gaussian prior. Summing over all ground atoms in WPLL is computationally expensive, so we only sum over a randomly-sampled fraction θ_{atoms} of them. We define the score of a clause c as $Score(c, D) = \log P^{\bullet}_{w',F',D}(X = x) - \pi l$, where $\log P^{\bullet}_{w',F',D}(X = x)$ is the WPLL, F' is a set containing c and one first-order unit clause for each predicate appearing in database D, and w' is a set of optimal weights for the clauses in F'.

We iterate over the clauses from shortest to longest. For each clause, we compare its scores against those of its sub-clauses (considered separately) that have already been retained. If the clause scores higher than all of these sub-clauses, it is retained; otherwise, it is discarded. In this manner, we discard clauses which are unlikely to be useful. Note that this process is efficient because the score of a clause only needs to be computed once and can be cached for future comparisons. (Alternatively, we could evaluate a clause against every combination of its sub-clauses, but this is computationally expensive because there could be many such combinations for long clauses.)

Finally we add the retained clauses to an MLN. We have the option of doing this in several ways. We could greedily add the clauses one at a time in order of decreasing score. After adding each clause, we relearn the weights and keep the clause in the MLN if it improves the overall WPLL. Alternatively, we could add all the clauses to the MLN, and learn weights using L1-regularization to

prune away 'bad' clauses by giving them zero weights [41]. Lastly, we could use L2-regularization instead if the number of clauses is not too large and rely on the regularization to give 'bad' clauses low weight. Optionally, to reduce the space of clauses considered, we discard clauses containing 'dangling' variables (i.e., variables which only appear once in a clause), since these are unlikely to be useful.

We use two heuristics to speed up clause evaluation. First we discard a path at the outset if it contains fewer than μ true ground atoms per hyperedge. This cuts the time we spend evaluating clauses that are not well supported by data. Second, before evaluating a clause's WPLL with respect to a database, we evaluate it with respect to the smaller number of ground atoms contained in the paths that gave rise to it. (Note that a clause can be created from different paths.) We limit the number of such ground atoms to a maximum of ν.

6.3 Experiments

6.3.1 Datasets

We carried out experiments to investigate whether LHL performs better than previous approaches and to evaluate the contributions of its components. We used three datasets publicly available at http://alchemy.cs.washington.edu. Their details are shown in Table 6.5.

Table 6.5: Information on datasets.

Dataset	Types	Constants	Predicates	True Atoms	Total Atoms
IMDB	4	316	6	1224	17,793
UW-CSE	9	929	12	2112	260,254
Cora	5	3079	10	42,558	687,422

IMDB. This dataset, created by Mihalkova and Mooney (2007) from the IMDB.com database, describes a movie domain. It contains predicates describing movies, actors, directors and their relationships (e.g, `Actor`(`person`), `WorkedIn`(`person`, `movie`), etc.) It is divided into 5 independent

folds. We omitted 4 equality predicates (e.g., SameMovie(movie, movie)) because they are superseded by the equality operator in the systems we are comparing.

UW-CSE. This dataset, prepared by Richardson and Domingos (2006) , describes an academic department. Its predicates describe students, faculty and their relationships (e.g, Professor(person), TaughtBy(course, person, quarter), etc.). The dataset is divided into 5 independent areas/folds (AI, graphics, etc.). We omitted 9 equality predicates for the same reason as above. (This dataset is different from that used for the MSL experiments in Section 3.6. Subsequent to those experiments, several people declined to have their information included in the dataset, so ground atoms describing them were removed.)

Cora. This dataset is a collection of citations to computer science papers, created by Andrew McCallum and later processed by Singla and Domingos (2006) into 5 folds for the task of deduplicating the citations, and their title, author and venue fields. Predicates include: SameCitation(cit1, cit2), TitleHasWord(title, word), etc. (This dataset is also different from that used for the MSL experiments in Section 3.6. It does not contain predicates which describe the percentage of words two fields have in common (e.g., CommonWordsInTitles80$-$100%(t, t′)). Such predicates are good indicators of whether two fields are the same, and removing them makes the deduplication task more challenging.)

6.3.2 Systems

We compared LHL to two state-of-the-art systems: BUSL [66] and Markov logic Structure Learner (MSL) [45] (described in Section 3.7 and Chapter 3 respectively). Both systems are implemented in the Alchemy software package [50].

To investigate the importance of hypergraph lifting, we removed the LiftGraph component from LHL and let FindPaths run on the unlifted hypergraph. The rules it learned were pruned by CreateMLN as per normal. We call this system LHL-FindPaths. We also investigated the contribution of hypergraph lifting alone by applying LiftGraph's MLN on the test sets. We call this system LHL-LiftGraph. We also investigated the effectiveness of the two heuristics in CreateMLN, by disabling them and observing the performance of the MLN thus learned by LHL. We call this system LHL-NoHeu. Altogether we compared six systems: LHL, LHL-NoHeu, LHL-FindPaths, LHL-

LiftGraph, BUSL and MSL. All systems are implemented in C++.

The following parameter values were used for the LHL systems on all datasets: $\lambda = 1$ (weight of exponential prior rule), $\mu = 50$, $\nu = 500$, $\theta_{atoms} = 0.5$. The other parameters were set as follows: ω =5 (IMDB, UW-CSE) and 4 (Cora); $\pi = 0.01$ (UW-CSE, Cora) and 0.1 (IMDB). (See Table 6.1 for parameter descriptions.) For BUSL and MSL, we set their parameters corresponding to π to values we used for LHL. We also set their $minWeight$ parameter to zero (empirically we found that this value performed better than their defaults). All other BUSL and MSL parameters were set to their default values. For all LHL systems, we used the following options:

- Created clauses with all combinations of negated/non-negated atoms in a variabilized path.
- Greedily added clauses one at a time in order of decreasing score to an initially empty MLN (as is done in BUSL).
- Excluded clauses with dangling variables from the final MLN to reduce the space of clauses considered. However, we did not impose this restriction on clauses with two or fewer literals because the number of such clauses is small. (To ensure fairness, we also tried excluding dangling clauses in BUSL and MSL, and report the best results for each.)

The parameters were set in an *ad hoc* manner, and per-fold optimization using a validation set could conceivably yield better results. All systems were run on identically configured machines (2.8GHz, 4GB RAM).

6.3.3 Methodology

For each dataset, we performed cross-validation using the five previously defined folds.

For IMDB and UW-CSE, we performed inference over the groundings of each predicate to compute their probabilities of being true, using the groundings of all other predicates as evidence. For Cora, we evaluated the systems' performances on the tasks of deduplicating the citations, and the title, author and venue fields. We ran inference over each of the four predicates `SameCitation`, `SameTitle`, `SameAuthor` and `SameVenue` in turn, using the groundings of all other predicates as

evidence. We used Alchemy's Gibbs sampling for all systems except LHL-LiftGraph. For LHL-LiftGraph, we used Alchemy's MC-SAT algorithm [77] because it has been shown to give better results for MLNs containing deterministic rules, which LHL-LiftGraph does. Each run of the inference algorithms drew 1 million samples, or ran for a maximum of 24 hours, whichever came earlier.

To evaluate the performance of the systems, we measured the average conditional log-likelihood of the test atoms (CLL) and the area under the precision-recall curve (AUC).

6.3.4 Results

Table 6.6 reports the AUCs, CLLs and runtimes. The AUC and CLL results are averages over all atoms in the test sets and their standard deviations. Runtimes are averages over the five folds.

We first compare LHL to BUSL and MSL. In both AUC and CLL, LHL outperforms BUSL and MSL on all datasets. The differences between LHL and BUSL/MSL on all datasets are statistically significant according to one-tailed paired t-tests (p-values ≤ 0.02 for both AUC and CLL). LHL is slower than BUSL and MSL on the smallest dataset (IMDB), mixed on the medium one (UW-CSE), and faster on the largest one (Cora). This suggests that LHL scales better than BUSL and MSL.

Next we compare LHL to its components LHL-LiftGraph and LHL-FindPaths. Comparing the runtimes of LHL and LHL-FindPaths, we see that LHL is much faster than LHL-FindPaths. LHL's AUC and CLL are similar to or better than LHL-FindPaths's on IMDB and UW-CSE, but are worse on Cora. These results suggest that: LHL is a lot faster than LHL-FindPaths without any loss in accuracy on some datasets; and when LHL-FindPaths does better, it does so at a huge computational cost (e.g., it took about 247 days to run on Cora[1]). LHL also outperforms LHL-LiftGraph on both AUC and CLL on the IMDB and UW-CSE datasets.[2] This suggests that LHL's ability to learn clauses that capture complex dependencies among predicates is an advantage over the simple rules in LHL-LiftGraphs.

Comparing LHL and LHL-NoHeu, we see that the two speedup heuristics in CreateMLN are effective in reducing LHL's runtime. On all datasets, we see that the heuristics do not compromise

[1]For each test fold, we ran FindPaths in parallel on all training folds and added the runtimes.

[2]LHL-LiftGraph on Cora crashed by running out of memory. Alchemy automatically converts the default atom prediction rules into clausal form and represents each clause separately, causing a blow-up in the number of clauses.

Table 6.6: Experimental results.

	IMDB			UW-CSE			Cora		
System	AUC	CLL	Time (min)	AUC	CLL	Time (hr)	AUC	CLL	Time (hr)
LHL	0.69±0.01	−0.13±0.00	15.63±1.88	0.22±0.01	−0.04±0.00	7.55±1.53	0.87±0.00	−0.26±0.00	14.82±1.78
LHL-NoHeu	0.69±0.01	−0.13±0.00	39.00±13.56	0.22±0.01	−0.04±0.00	158.24±46.70	0.87±0.00	−0.26±0.00	33.99±3.86
LHL-FindPaths	0.69±0.01	−0.13±0.00	242.41±30.31	0.19±0.01	−0.04±0.00	56.69±19.98	0.91±0.00	−0.17±0.00	5935.50±39.21
LHL-LiftGraph	0.45±0.01	−0.27±0.01	0.18±0.01	0.14±0.01	−0.06±0.00	0.001±0.000	-	-	0.01±0.01
BUSL	0.47±0.01	−0.14±0.00	4.69±1.02	0.21±0.01	−0.05±0.00	12.97±9.80	0.17±0.00	−0.37±0.00	18.65±9.52
MSL	0.41±0.01	−0.17±0.00	2.79±0.59	0.18±0.01	−0.57±0.00	2.13±0.38	0.17±0.00	−0.37±0.00	65.60±1.82

the quality of the MLNs that LHL learns because LHL and LHL-NoHeu have the same AUC and CLL. Examining the runtime of LiftGraph, we found that it accounts for only a tiny fraction of LHL runtime (less than 0.1%).

The results for MSL on UW-CSE and Cora are not directly comparable to those reported in Section 3.6 for the following reasons. First, as described in Section 6.3.1, the datasets are not exactly the same. Second, the experiments in Section 3.6 evaluated MSL by computing the probability that a ground atom is true given all other ground atoms as evidence, a much easier task than the evaluation used in this chapter. Third, we did not use any domain-specific declarative bias to guide clause construction as was previously done. (Notice how LHL is able to overcome the myopia of greedy search without the help of this bias.)

The results for BUSL on IMDB and UW-CSE are also different from those reported by Mihalkova and Mooney (2007) . Unlike them, we omitted evaluating the equality predicates (as mentioned earlier) because they are superfluous. This reason also contributes to the difference in MSL's performance.

The following are examples of (weighted) rules learned by LHL.

- $\texttt{Director}(\texttt{d}) \wedge \texttt{Actor}(\texttt{a}) \wedge \texttt{InMovie}(\texttt{d},\texttt{m}) \wedge \texttt{InMovie}(\texttt{a},\texttt{m}) \Rightarrow \texttt{WorkedUnder}(\texttt{a},\texttt{d})$. (If a director d and an actor a work in the same movie, then a is likely to work under d's direction.)

- $\texttt{HasFacultyPosition}(\texttt{s},\texttt{p}) \Rightarrow \neg\texttt{Student}(\texttt{s})$. (If a person s has a faculty position p, then she is not a student).

- $\texttt{TitleHasWord}(\texttt{t},\texttt{w}) \wedge \texttt{TitleHasWord}(\texttt{t}',\texttt{w}) \wedge \texttt{SameTitle}(\texttt{t},\texttt{t}')$. (The actual rule learned by LHL is the negation of the above rule (i.e., a clause) with negative weight $-w$. Under Markov logic, it is equivalent to the above conjunction with positive weight w. The rule models the condition that two titles t and t' that contain the same word w actually refer to the same title.)

- $\texttt{AuthorOfCit}(\texttt{a},\texttt{c}) \wedge \texttt{AuthorOfCit}(\texttt{a}',\texttt{c}') \wedge \texttt{SameAuthor}(\texttt{a},\texttt{a}') \Rightarrow \texttt{SameCitation}(\texttt{c},\texttt{c}')$. (If two citations c and $\texttt{c}'$ respectively have authors a and $\texttt{a}'$ that refer to the same author, then c and $\texttt{c}'$ refer to the same citation.)

6.4 Related Work

Besides relational pathfinding [84], ILP approaches with bottom-up aspects include Muggleton & Buntine (1988), Muggleton & Feng (1990), etc. These approaches are vulnerable to noise in the data and only create clauses to predict a single target predicate.

Popescul and Ungar (2004) have also used clustering to improve probabilistic rule induction. Their approach is limited to logistic regression and SQL rules, uses a very simple clustering method (k-means), and requires pre-specifying the number of clusters. Craven and Slattery (2001) learn first-order rules for hypertext classification using naive Bayes models as invented predicates.

The idea of lifting comes from theorem-proving in first-order logic. In recent years, it has been extended to inference in MLNs and other probabilistic languages. In lifted belief propagation [96], the algorithm forms clusters of ground atoms and clusters of ground clauses. It performs inference over the more compact network of clusters, thereby improving efficiency. This is analogous to LHL's approach of forming clusters of ground atoms to create a lifted hypergraph in which the search for clauses is more efficient.

6.5 Conclusion

In this chapter, we proposed LHL, a novel algorithm that uses the data to constrain the space of candidate clauses to overcome the limitations of prior top-down systems such as MSL. LHL lifts the training data into a compact hypergraph by clustering constants into high-level concepts and uses relational pathfinding over the hypergraph to find clauses. Our empirical results show the efficacy of LHL.

Even though LHL ameliorates the cost of relational pathfinding by lifting a ground hypergraph, it still incurs a large computational cost when finding long paths (beyond 4-5 literals) over the entire lifted hypergraph. In the next chapter, we present a solution to this problem that constrains the search for long clauses to within recurring patterns in the ground hypergraph.

Chapter 7

LEARNING MARKOV LOGIC NETWORKS USING STRUCTURAL MOTIFS

7.1 Introduction

In Chapters 3 and 6, we described several MLN structure learners, all of which are only able to learn short clauses (4-5 literals) due to the extreme computational cost of learning. In this chapter, we address this problem with LSM [49], the first MLN structure learner capable of efficiently and accurately learning long clauses. Its key insight is that relational data usually contains recurring patterns, which we term *structural motifs*. These motifs confer three benefits. First, by confining its search to *within* motifs, LSM need not waste time following spurious paths *between* motifs. Second, LSM only searches in each unique motif once, rather than in all its occurrences in the data. Third, by creating various motifs over a set of objects, LSM can capture different interactions among them. A structural motif is frequently characterized by objects that are densely connected via many paths, allowing us to identify motifs using the concept of *truncated hitting time* [91] in *random walks* [59]. This concept has been used in many applications, and we are the first to successfully apply it to learning MLN formulas.

The remainder of this chapter is organized as follows. We begin by reviewing some background on truncated hitting time and random walks in the next section. Then we describe LSM in detail (Section 7.3), report our experiments (Section 7.4), and discuss related work (Section 7.5).

7.2 Random Walks and Hitting Times

Random walks and hitting times are defined in terms of *hypergraphs*. A hypergraph is a straightforward generalization of a graph in which an edge can link any number of nodes, rather than just two. Formally, a hypergraph G is a pair (V, E) where V is a set of nodes, and E is a set of labeled non-empty subsets of V called hyperedges. A *path* of length t between nodes u and u' is an alternating sequence of nodes and hyperedges $(v_0, e_0, v_1, e_1, \ldots, e_{t-1}, v_t)$ such that $u = v_0$, $u' = v_t$, $e_i \in E$, $v_i \in e_i$ and $v_{i+1} \in e_i$ for $i \in \{0, \ldots, t\}$. u is said to be *reachable* from u' iff there is path from u to

u'. G is *connected* if all its nodes are reachable from each other. p_s^v denotes a path from s to v.

In a *random walk* [59], we travel from node to node via hyperedges. Suppose that at some time step, we are at node i. In the next step, we move to one of its neighbors j by first randomly choosing a hyperedge e from the set E_i of hyperedges that are incident to i, and then randomly choosing j from among the nodes that are connected by e (excluding i). The probability of moving from i to j is called the *transition probability* p_{ij}, and is given by $p_{ij} = \sum_{e \in E_i \cap E_j} \frac{1}{|E_i|} \frac{1}{|e|-1}$. Note that the transition probabilities from a node to all its neighbors sum to 1.

The *hitting time* h_{ij} [1] from node i to j is defined as the average number of steps one takes in a random walk starting from i to visit j for the first time. It is recursively defined as $h_{ij} = 1 + \sum_k p_{ik} h_{kj}$ if $i \neq j$ and zero otherwise. The larger the number of paths between i and j, and the shorter the paths, the smaller the hitting time. Thus, hitting time is useful for capturing the notion of 'closeness' between nodes. However, computing the hitting times between all pairs of nodes require at least $O(|V|^2)$ time, and thus is intractable for large hypergraphs.

To circumvent this problem, Sarkar et al. (2008) introduced the notion of *truncated* hitting time. The truncated hitting time h_{ij}^T from node i to j is defined as the average number of steps one takes to reach j for the first time starting from i in a random walk that is *limited to at most T steps*. $h_{ij}^T = 0$ if $i = j$ or $T = 0$, and $h_{ij}^T = T$ if j is not reach in T steps. Thus, starting from a node i, we only need to compute the time it takes to reach nodes in its vicinity, rather than all nodes. As $T \to \infty, h_{ij}^T \to h_{ij}$. Sarkar et al. showed that truncated hitting time can be approximated accurately with high probability by sampling. They run W independent length-T random walks from node i. In w of these runs, node j is visited for the first time at time steps $t_j^1, \ldots, t_j^w$. The *estimated* truncated hitting time is given by

$$\hat{h}_{ij}^T = \frac{\sum_{k=1}^{w} t_j^k}{W} + (1 - \frac{w}{W})T. \tag{7.1}$$

They also showed that the number of samples of random walks W has to be at least $\frac{1}{2\epsilon^2} \log \frac{2|V|}{d}$ in order for the estimated truncated hitting time to be a good estimate of the actual truncated hitting time with high probability, i.e., for $P(|\hat{h}_{ij}^T - h_{ij}^T| \leq \epsilon T) \geq 1 - \delta$, where ϵ and δ are user-specified parameters, and $0 \leq \epsilon, \delta \leq 1$.

7.3 Learning via Structural Motifs

We call our algorithm Learning using Structural Motifs (LSM; Table 7.1). The crux of LSM is that relational data frequently contains recurring patterns of densely connected objects, and by limiting our search to within these patterns, we can find good rules quickly. We call such patterns *structural motifs*.

A structural motif is a set of literals, which defines a set of clauses that can be created by forming disjunctions over the negations/non-negations of one or more of the literals. Thus, it defines a subspace within the space of all clauses. LSM discovers subspaces where literals are densely connected and groups them into a motif. To do so, LSM views a database as a hypergraph with constants as nodes, and true ground atoms as hyperedges. Each hyperedge is labeled with a predicate symbol. LSM groups nodes that are densely connected by many paths, and the hyperedges connecting them into a motif. Then it compresses the motif by clustering nodes into high-level concepts, reducing the search space of clauses in the motif. Next it quickly estimates whether the motif appears often enough in the data to be retained. Finally, LSM runs relational pathfinding on each motif to find candidate rules and retains the good ones in an MLN.

Figure 7.1 provides an example of a graph created from a university database describing two departments. The bottom motifs are extracted from the top graph. Note that the motifs have gotten rid of the spurious link between departments, preventing us from tracing paths straddling departments that do not translate to good rules. Also note that by searching only once in each unique motif, we avoid duplicating the search in all its occurrences in the graph. Observe that both motifs are created from each department's subgraph. In the left motif, individual students and books are respectively clustered into high-level concepts *Student* and *Book* because they are indistinguishable with respect to professor $P1$ (they have symmetrical paths from $P1$). In the right motif, the clustering is done with respect to book $B1$. LSM's ability to create different motifs over a set of objects allows it to capture various interactions among the objects, and thus to potentially discover more good rules.

LSM differs from LHL (Chapter 6) in the following ways. First, LHL finds a single clustering of nodes, but in LSM, a node can belong to different clusters. Second, in LHL, two nodes v and v' are clustered together if they are related to many common nodes. Thus, intuitively, LHL is making use of length-2 paths to determine the similarity of nodes (e.g., $vrwr'v'$ where v and v' are connected

Table 7.1: LSM algorithm.

Input: $G = (V, E)$, a ground hypergraph representing a database

Output: MLN, a set of weighted clauses

$Motifs \leftarrow \emptyset$
For each $s \in V$
  Run N_{walks} random walks of length T from s to estimate h_{sv}^T for all $v \in V$
  Create V_s to contain nodes whose $h_{sv}^T < \theta_{hit}$
  Create E_s to contain hyperedges that only connect to V_s
  Partition V_s into $\{A_1, \ldots A_l\}$ where $\forall v \in A_j, \exists v' \in A_j : |h_{sv}^T - h_{sv'}^T| < \theta_{sym}$
  $\mathcal{V}_s \leftarrow \emptyset$
  For each $A_i \in \{A_1, \ldots A_l\}$
    Partition A_i into $H = \{H_1, \ldots, H_m\}$ so that symmetrical nodes in A_i belong to the same $H_j \in H$
    Add $H_1, \ldots, H_m$ to $\mathcal{V}_s$
  Create $\mathcal{E}_s = \{E_1, \ldots, E_k\}$ where hyperedges in E with the same label, and that connect to the same sets in $\mathcal{V}_s$ belong to the same $E_j \in \mathcal{E}_s$.
  Let lifted hypergraph $L = (\mathcal{V}_s, \mathcal{E}_s)$
  Create $Motif(L)$ using DFS, add it to $Motifs$
For each $m \in Motifs$
  Let n_m be the number of unique true groundings returned by DFS for m
  If $n_m < \theta_{motif}$, remove m from $Motifs$
$Paths \leftarrow$ FindPaths($Motifs$)
$MLN \leftarrow$ CreateMLN($Paths$)
Return MLN

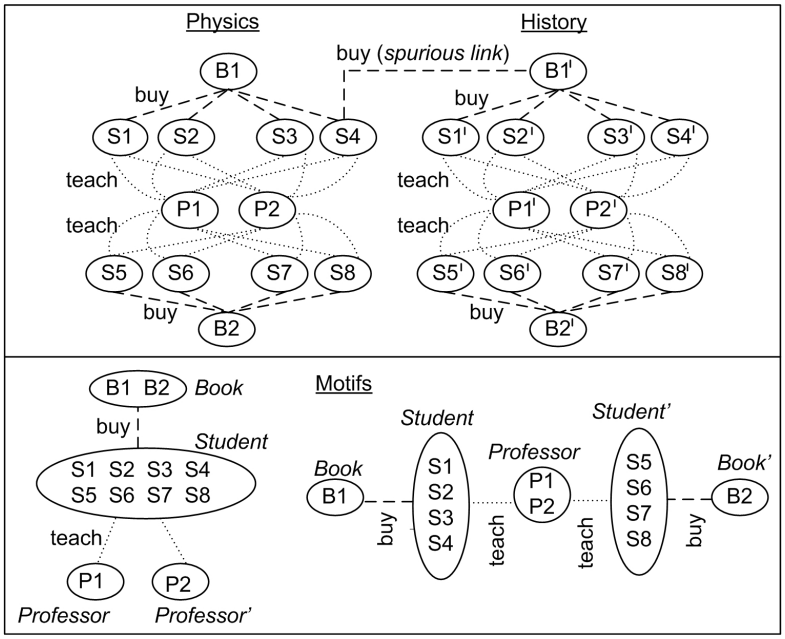

Figure 7.1: Motifs extracted from a ground hypergraph.

to w via edges r and r'). In contrast, LSM uses longer paths, and thus more information, to find various clusterings of nodes. Third, spurious edges present in LHL's initial ground hypergraph are retained in the lifted one, but these edges are ignored by LSM.

7.3.1 Preliminaries

We define some terms and state a proposition. (The proofs of all propositions are given in Appendix E.) A *ground* hypergraph $G = (V, E)$ has constants as nodes, and true ground atoms as hyperedges. An r-hyperedge is a hyperedge labeled with predicate symbol r. There cannot be two or more r-hyperedges connected to a set of nodes because they correspond to the same ground atom.

$\sigma(p)$ refers to the string that is created by replacing nodes in path p with integers indicating the order in which the nodes are first visited, and replacing hyperedges with their predicate symbols. Nodes which are visited simultaneously via a hyperedge have their order determined by their argument positions in the hyperedge. Two paths p and p' are *symmetrical* iff $\sigma(p) = \sigma(p')$.

Nodes v and v' are symmetrical relative to s, denoted as $Sym_s(v, v')$, iff there is a bijective mapping between the set of all paths from s to v and the set of all paths from s to v' such that

each pair of mapped paths are symmetrical. Node sets $V = \{v_1, \ldots, v_n\}$ and $V' = \{v'_1, \ldots, v'_n\}$ are symmetrical iff $Sym_s(v_i, v'_i)$ for $i = 1, \ldots, n$. Note that Sym_s is reflexive, symmetric and transitive.

Note that symmetrical nodes v and v' have identical truncated hitting times from s (since every path of v is symmetrical to some path of v' and vice versa). Also note that symmetrical paths p_s^v and $p_s^{v'}$ have the same probability of being sampled respectively from the set of all paths from s to v and the set of all paths from s to v'.

$L_{G,s}$ is the 'lifted' hypergraph that is created as follows from a ground hypergraph $G = (V, E)$ whose nodes are all reachable from a node s. Partition V into disjoint subsets $\mathcal{V} = \{V_1, \ldots, V_k\}$ (i.e., $V_i \neq \emptyset$, $\cup_{i=0}^{k} V_i = V$, and $V_i \cap V_j = \emptyset$ for $i \neq j$) such that all nodes with symmetrical paths from s are in the same V_i. Partition E into disjoint subsets $\mathcal{E} = \{E_1, \ldots, E_l\}$ such that all r-hyperedges that connect nodes from the same V_i's are grouped into the same E_j, which is also labeled r. $L_{G,s} = (\mathcal{V}, \mathcal{E})$ intuitively represents a high-level concept with each V_i, and an interaction between the concepts with each E_j. Note that $L_{G,s}$ is connected since no hyperedge in E is removed during its construction. Also note that s is in its own $V_s \in \mathcal{V}$ since no other node has the empty path to it.

Proposition 1. *Let v, v' and s be nodes in a ground hypergraph whose nodes are all reachable from s, and $Sym_s(v, v')$. If an r-hyperedge connects v to a node set W, then an r-hyperedge connects v' to a node set W' that is symmetrical to W.*

We create a structural motif $Motif(L_{G,s})$ from $L_{G,s} = (\mathcal{V}, \mathcal{E})$ as follows. We run depth-first search (DFS) on $L_{G,s}$ but treat hyperedges as nodes and vice versa (a straightforward modification), allowing DFS to visit each hyperedge in $\mathcal{E}$ exactly once. (The DFS pseudocode is given in Table 7.2.) DFS starts from an $E_j \in \mathcal{E}$. Whenever it visits a hyperedge $E_j \in \mathcal{E}$, DFS selects an $e_j \in E_j$ that is connected to a ground node $v_i \in V$ that is linked to the e_i selected in the previous step (e_j exists by Proposition 1). When several e_j's are connected to v_i, it selects the one connected to the smallest number of unique nodes (so as to create the simplest motif with the smallest number of variables). The selected e_j's are then variabilized (the same variable is used for the same node across e_j's), and added as literals to the set $Motif(L_{G,s})$. Let $Conj(m)$ denote the conjunction formed by conjoining the (positive) literals in motif m. Note that the selected e_j's are connected, and form a true grounding of $Conj(Motif(L_{G,s}))$. (The true grounding will be used later to estimate the total

number of true groundings of $Conj(Motif(L_{G,s}))$ in the data.) Also note that since DFS visits every E_j once (in particular the one connected to V_s), s is in the true grounding.

7.3.2 Motif Identification

LSM begins by creating a ground hypergraph from a database. Then it iterates over the nodes. For each node i, LSM finds nodes that are symmetrical relative to i. To do so, it has to compare all paths from i to all other nodes, which is intractable. Thus LSM uses an approximation. It runs N_{walks} random walks of length T from i (line 3 of Table 7.1). In each random walk, when a node is visited, the node stores the path p to it as $\sigma(p)$ (up to a maximum of Max_{paths} paths), and records the number of times $\sigma(p)$ is seen. After running all random walks, LSM estimates the truncated hitting time h_{iv}^T from i to each node v that is visited at least once using Equation 7.1. (Nodes not visited have $h_{iv}^T = T$.) Nodes whose h_{iv}^T's exceed a threshold $\theta_{hit} < T$ are discarded (these are 'too loosely' connected to i). The remaining nodes and the hyperedges that only connect to them constitute a ground hypergraph G (lines 4-5). LSM groups together nodes in G whose h_{iv}^T's are less than θ_{sym} apart as potential symmetrical nodes (line 6). (In a group, a node only needs to have similar h_{iv}^T with at least one other node.)

Within each group, LSM uses greedy agglomerative clustering to cluster symmetrical nodes together (lines 8-11). Two nodes are approximated as symmetrical if their distributions of stored paths are similar. Since the most frequently appearing paths are more representative of a distribution, we only use the top N_{top} paths in each node. Path similarity is measured using Jensen-Shannon divergence (Fugledge & Topsoe, 2004; a symmetric version of the Kullback-Leibler divergence). Each node starts in its own cluster. At each step, LSM merges the pair of clusters whose path distributions are most similar. When there is more than one node in a cluster, its path distribution is the average over those of its nodes. The clustering stops when no pair of clusters have divergence less than θ_{js}. Once the clusters of symmetrical nodes are identified, LSM creates lifted hypergraph $L_{G,s}$ and motif $Motif(L_{G,s})$ using DFS as described earlier (lines 12-13). Then LSM repeats the process for the next node $i + 1$.

After iterating over all nodes, LSM will have created a set of motifs. It then estimates how often a motif m appears in the data by computing a lower bound n_m on the number of true groundings

Table 7.2: DFS algorithm

Input: $L_{G,s} = (\mathcal{V}, \mathcal{E})$, a lifted hypergraph

Output: $Motif(L_{G,s})$, a motif

$E' \leftarrow \emptyset$

$V' \leftarrow \emptyset$

For each edge set $E \in \mathcal{E}$

  Mark E as `NotVisited`

Pick an edge set $E_i \in \mathcal{E}$

Pick an edge $e_i \in E_i$

Enqueue $\{(E_i, e_i)\}$ into $Queue$

Mark E_i as `Visited`

While $Queue \neq \emptyset$

  Dequeue (E, e) from head of $Queue$

  $E' \leftarrow E' \cup \{e\}$

  For each node v connected by e

    $V' \leftarrow V' \cup \{v\}$

    Let $V_v \in \mathcal{V}$ be the node set containing v

    For each edge set $E_j \in \mathcal{E}$ incident to V_v

      If E_j is `NotVisited`

        Pick an edge $e_j \in E_j$ that is incident to v (an e_j exists by Proposition 1)

        Enqueue $\{(E_j, e_j)\}$ into $Queue$

        Mark E_j as `Visited`

Add edges in E' as literals to $Motif(L_{G,s})$

Return $Motif(L_{G,s})$

---

of $Conj(m)$. It sets n_m to the number of unique true groundings of m that are returned by the DFS algorithm. If n_m is less than a threshold θ_{motif}, the motif is discarded (lines 14-16). Finally, a retained motif that is a sub-conjunction of another is discarded because the larger motif contains all rules that can be found in the smaller one.

Our algorithm can be viewed as a search for motifs that maximizes an upper bound on the log posterior of the data, $\log P(W, C|X) \propto \log P(X|W, C) + \log P(W|C) + \log P(C)$ where X is a database of ground atoms, C is the set of rules in an MLN, W is their corresponding weights, and $P(X|W, C)$ is given by Equation 2.3. We define the prior on C as $P(C) = \exp(-|C|)$. To constrain our search space, we restrict C to be conjunctions of positive literals (without loss of generality [101]). We also impose a zero-mean Gaussian prior on each weight, so $\log P(W|C)$ is concave. Since both $\log P(X|W, C)$ and $\log P(W|C)$ are concave, their sum is also concave and hence has a global maximum. Let $L_{W,C}(X) = \log P(X|W, C) + \log P(W|C)$.

Proposition 2. *The maximum value of $L_{W,C}(X)$ is attained at $W = W_0$ and $C = C_0$ where C_0 is the set of all possible conjunctions of positive ground literals that are true in X, and W_0 is the set containing the globally optimal weights of the conjunctions.*

Let C' be the set of ground conjunctions obtained by replacing the true groundings in C_0 of a first-order conjunction c with c. Let W' be the optimal weights of C'. The difference in log posterior for (W', C') and (W_0, C_0) is given by $\Delta = L_{W',C'}(X) - L_{W_0,C_0}(X) + (\log P(C') - \log P(C_0))$. Using Proposition 2, we know that $L_{W',C'}(X) - L_{W_0,C_0}(X) \leq 0$. Thus, $\Delta \leq \log P(C') - \log P(C_0) = n_c - 1$, where n_c is the number of true groundings of c. Since Δ is upper-bounded by $n_c - 1$, we want to find motifs with large n_c. We do so by requiring the motifs to have $n_c \geq \theta_{motif}$.

7.3.3 PathFinding and MLN Creation

LSM uses the FindPaths and CreateMLN components of the LHL system (Chapter 6).

LSM finds paths in each identified motif in the same manner as LHL's FindPaths. The paths are limited to a user-specified maximum length.

After that, LSM creates candidate clauses from each path in a similar way as LHL's CreateMLN, with some modifications. At the start of CreateMLN, LSM counts the true groundings of all possible unit and binary clauses (i.e., clauses with one and two literals) to find those that are always true or

always false in the data. (Since the number of predicates is usually small, this is not a computational burden.) It then removes every candidate clause that contains unit/binary sub-clauses that are always true because it is always satisfied. If a candidate clause c contains unit/binary sub-clauses that always false, and if the clause c' formed by removing the unit/binary sub-clauses is also a candidate clause, then c is removed because it is a duplicate of c'. LSM also detects whether a binary predicate R is symmetric by evaluating whether $R(x, y) \Leftrightarrow R(y, x)$ is always true. LSM then removes clauses that are identical modulo the order of variables in symmetric binary predicates. These changes speed up CreateMLN by reducing the number of candidates. At the end of CreateMLN, rather than adding clauses greedily to an empty MLN (which is susceptible to local optima), LSM adds all clauses to the MLN, finds their optimal weights, and removes those whose weights are less than θ_{wt}. (We use a zero-mean Gaussian prior on each weight.) LSM also adds a heuristic to speed up CreateMLN. Before evaluating the WPLLs of candidate clauses against the data, it evaluates them against the ground hypergraphs that give rise to the motifs where the candidate clauses are found. Since such ground hypergraphs contain fewer atoms, it is faster to evaluate against them to quickly prune bad candidates.

In CreateMLN, LSM evaluates each candidate clause using WPLL (Equation 3.1) as in LHL. Summing over all ground atoms in WPLL is computationally expensive, so we only sum over a randomly-sampled fraction θ_{atoms} of them.

7.4 Experiments

Our experiments used the IMDB, UW-CSE and Cora datasets as described in Section 6.3.

7.4.1 Systems

We compared LSM to three state-of-the-art systems: LHL, BUSL and MSL. We implemented LHL and used the BUSL and MSL implementations in the Alchemy software package [50]. In LHL, we used the modified CreateMLN because it was faster. In BUSL, rather than adding clauses to an MLN greedily (which was susceptible to local optima), we adopted LSM's approach of adding all clauses to the MLN, finding their optimal weights (using a zero-mean Gaussian prior on each weight), and removing those whose weights were less than θ_{wt}.

We ran each system with two limits on clause length. The short limit was set to 5 (IMDB, UW-CSE) and 4 (Cora) as in experiments with LHL (Section 6.3). The long limit was set to 10. Systems with the short and long limits are respectively appended with '-S' and '-L'. For the short limit, we allowed LSM, LHL and BUSL to create more candidate clauses from a candidate containing only negative literals by non-negating the literals in all possible ways. For the long limit, we permitted a maximum of two non-negations to avoid generating too many candidates. As in Section 6.3, we disallowed clauses with variables that only appeared once to reduce the space of clauses considered. However, we did not impose this restriction on clauses with two or fewer literals because the number of such clauses was small.

To investigate the individual contributions of our motif identification algorithm and the heuristic in CreateMLN, we removed them to give the respective systems LSM-NoMot and LSM-NoHeu. LSM-NoMot found paths directly on the ground hypergraph created from a database. Altogether, we compared twelve systems.

The LSM parameter values were: $N_{walks} = 15,000$, $\epsilon = 0.1$, $\delta = 0.05$, $T = 5$, $\theta_{hit} = 4.9$, $\theta_{sym} = 0.1$, $\theta_{js} = 1$, $N_{top} = 3$, $Max_{paths} = 100$, $\theta_{motif} = 10$, $\pi = 0.1$ (IMDB) and 0.01 (UW-CSE, Cora), $\theta_{atoms} = 0.5$, $\theta_{wt} = 0.01$. (See Section 7.3.2 and 7.3.3 for parameter descriptions.) The other systems had their corresponding parameters set to the same values, and their other parameters set to default values.

The parameters were set in an *ad-hoc* manner, and per-fold optimization using a validation set could conceivably yield better results. All systems were run on identically configured machines (2.3GHz, 16GB RAM, 4096KB CPU cache) for a maximum of 28 days.

7.4.2 Methodology

For each dataset, we performed cross-validation using the five folds in each dataset. For IMDB and UW-CSE, we performed inference over the groundings of each predicate to compute their probabilities of being true, using the groundings of all other predicates as evidence. For Cora, we ran inference over each of the four predicates `SameCitation`, `SameTitle`, `SameAuthor` and `SameVenue` in turn, using the groundings of all other predicates as evidence. We denote this task as "Cora (One Predicate)" to differentiate it from the next task. We also ran inference over the groundings of all

four predicates together, which is a more challenging task than inferring the groundings of each individually. We denote this task as "Cora (Four Predicates)". For this task, we split each test fold into 5 sets by randomly assigning each paper and its associated ground atoms to a set. We had to run inference over each test set separately in order for the inference algorithm to work within the available memory. To obtain the best possible results for an MLN, we relearned its clause weights for each query predicate (or set of query predicates in the case of Cora) before performing inference. This accounts for the differences in our results from those reported for LHL's experiments (Section 6.3). We used Alchemy's Gibbs sampling for all systems. Each run of the inference algorithms drew 1 million samples, or ran for a maximum of 24 hours, whichever came earlier. To evaluate the performance of the systems, we measured the average conditional log-likelihood of the test atoms (CLL) and the area under the precision-recall curve (AUC).

7.4.3 Results

Tables 7.3 and 7.4 report AUCs, CLLs and runtimes. The AUC and CLL results are averages over all atoms in the test sets and their standard deviations. Runtimes are averages over the five folds.

We first compare LSM to LHL. The results indicate that LSM scales better than LHL, and that LSM equals LHL's predictive performance on small simple domains, but surpasses LHL on large complex ones. LSM-S is marginally slower than LHL-S on the smallest dataset, but is faster on the two larger ones. The scalability of LSM becomes clear when the systems learn long clauses: LSM-L is consistently 100-100,000 times faster than LHL-L on all datasets.[1] Note that LSM-L performs better than LSM-S on both AUC and CLL on Cora (Four Predicates), substantiating the importance of learning long rules. LSM-S and LSM-L learned the same MLNs on IMDB and UW-CSE, and thus have the same AUC and CLL. Even though learning longer rules did not improve peformance on IMDB and UW-CSE, it is still useful to be able to efficiently explore their spaces of long rules to ascertain a good rule length when we have no *a priori* knowledge about their domains.

We next compare LSM to MSL and BUSL. LSM consistently outperforms MSL on AUC and CLL for both short and long rules; and draws with BUSL on UW-CSE, but does better on IMDB and Cora. In terms of runtime, the results are mixed. Observe that BUSL and MSL have similar runtimes

[1] LHL-L on UW-CSE and Cora, and LSM-NoMot-L exceeded the time bound of 28 days. We estimated their runtimes by extrapolating from the number of atoms they had initiated their search from.

Table 7.3: Area under precision-recall curve (AUC) and conditional log-likelihood (CLL) of test atoms.

	IMDB		UW-CSE		Cora (One Predicate)		Cora (Four Predicates)	
System	AUC	CLL	AUC	CLL	AUC	CLL	AUC	CLL
LSM-S	0.71±0.01	−0.06±0.00	0.22±0.01	−0.03±0.00	0.98±0.00	−0.02±0.00	0.92±0.00	−0.42±0.00
LSM-L	0.71±0.01	−0.06±0.00	0.22±0.01	−0.03±0.00	0.98±0.00	−0.02±0.00	0.97±0.00	−0.23±0.00
LSM-NoHeu-S	0.71±0.01	−0.06±0.00	0.22±0.01	−0.03±0.00	0.98±0.00	−0.02±0.00	0.93±0.00	−0.39±0.00
LSM-NoHeu-L	0.71±0.01	−0.06±0.00	0.22±0.01	−0.03±0.00	0.98±0.00	−0.02±0.00	0.97±0.00	−0.23±0.00
LSM-NoMot-S	0.71±0.01	−0.06±0.00	0.23±0.01	−0.03±0.00	0.98±0.00	−0.02±0.00	0.93±0.00	−0.38±0.00
LSM-NoMot-L	0.34±0.01	−0.18±0.00	0.13±0.01	−0.04±0.00	0.57±0.00	−0.29±0.00	0.47±0.00	−0.94±0.00
LHL-S	0.71±0.01	−0.06±0.00	0.21±0.01	−0.03±0.00	0.95±0.00	−0.04±0.00	0.76±0.00	−0.88±0.00
LHL-L	0.71±0.01	−0.06±0.00	0.13±0.01	−0.04±0.00	0.57±0.00	−0.29±0.00	0.47±0.00	−0.94±0.00
BUSL-S	0.48±0.01	−0.11±0.00	0.22±0.01	−0.03±0.00	0.57±0.00	−0.29±0.00	0.47±0.00	−0.94±0.00
BUSL-L	0.48±0.01	−0.11±0.00	0.22±0.01	−0.03±0.00	0.57±0.00	−0.29±0.00	0.47±0.00	−0.94±0.00
MSL-S	0.38±0.01	−0.17±0.00	0.19±0.01	−0.04±0.00	0.57±0.00	−0.29±0.00	0.47±0.00	−0.94±0.00
MSL-L	0.38±0.01	−0.17±0.00	0.18±0.01	−0.04±0.00	0.57±0.00	−0.29±0.00	0.47±0.00	−0.94±0.00

Table 7.4: System runtimes. The times for Cora (One Predicate) and Cora (Four Predicates) are the same.

System	IMDB (hr)	UW-CSE (hr)	Cora (hr)
LSM-S	0.21±0.02	1.38±0.3	1.33±0.03
LSM-L	0.31±0.04	4.52±2.35	20.57±7.29
LSM-NoHeu-S	0.13±0.03	10.01±5.06	1.7±0.05
LSM-NoHeu-L	0.29±0.09	13.4±6.11	48.56±16.06
LSM-NoMot-S	1.09±0.22	50.83±18.33	332.82±60.54
LSM-NoMot-L	160,000±12,000	280,000±35,000	5,700,000±10^5
LHL-S	0.18±0.02	5.29±0.81	1.92±0.02
LHL-L	73.45±11.71	120,000±13,000	230,000±7000
BUSL-S	0.03±0.01	2.77±1.06	1.83±0.04
BUSL-L	0.03±0.01	2.77±1.06	1.83±0.04
MSL-S	0.02±0.01	1.07±0.21	9.96±1.59
MSL-L	0.02±0.01	26.22±26.14	9.81±1.50

when learning both short and long rules (with the exception of MSL-L on UW-CSE). Tracing the steps taken by BUSL and MSL, we found that the systems took the same greedy search steps when learning both short and long rules, thus resulting in the same locally optimal MLNs containing only short rules. In contrast, LSM-L found longer rules than LSM-S for all datasets, even though these were only retained by CreateMLN for Cora.

Comparing LSM to LSM-NoHeu, we see that LSM's heuristic is effective in speeding it up. An exception is LSM-NoHeu on IMDB. This is not surprising because the small size of IMDB allows candidate clauses to be evaluated quickly against the database, obviating the need for heuristics. This suggests that the heuristic should only be employed on large datasets. Note that even though removing the heuristic improved LSM-S's performance on Cora (Four Predicates) by 1% on AUC and by 7% on CLL, the improvements are achieved at a great cost of 28% increase in runtime. Comparing LSM to LSM-NoMot, we see the importance of motifs in making LSM tractable.

Our runtimes are faster than those reported in LHL's experiments because of our modifications to CreateMLN, and our machines are better configured (4 times more RAM, 8 times more CPU cache).

The following are examples of long (weighted) rules learned by LSM on Cora

- $\texttt{AuthorOfCit}(\mathtt{a}, \mathtt{c}) \wedge \texttt{AuthorOfCit}(\mathtt{a}', \mathtt{c}') \wedge \texttt{SameAuthor}(\mathtt{a}, \mathtt{a}') \wedge \texttt{TitleOfCit}(\mathtt{t}, \mathtt{c}) \wedge \texttt{TitleOfCit}(\mathtt{t}', \mathtt{c}') \wedge \texttt{SameTitle}(\mathtt{t}, \mathtt{t}') \Rightarrow \texttt{SameCitation}(\mathtt{c}, \mathtt{c}')$. (If two citations $\mathtt{c}$ and $\mathtt{c}'$ have the same author and title, then they are likely to be the same citation.)

- $\texttt{VenueOfCit}(\mathtt{v}, \mathtt{c}) \wedge \texttt{VenueOfCit}(\mathtt{v}, \mathtt{c}') \wedge \texttt{AuthorOfCit}(\mathtt{a}, \mathtt{c}) \wedge \texttt{AuthorOfCit}(\mathtt{a}', \mathtt{c}') \wedge \texttt{SameAuthor}(\mathtt{a}, \mathtt{a}') \wedge \texttt{TitleOfCit}(\mathtt{t}, \mathtt{c}) \wedge \texttt{TitleOfCit}(\mathtt{t}', \mathtt{c}') \Rightarrow \texttt{SameTitle}(\mathtt{t}, \mathtt{t}')$. (If two citations $\mathtt{c}$ and $\mathtt{c}'$ have identical venues and the same author, then they are likely to have the same title.)

- $\texttt{AuthorHasWord}(\mathtt{a}, \mathtt{w}) \wedge \texttt{AuthorHasWord}(\mathtt{a}', \mathtt{w}') \wedge \texttt{AuthorHasWord}(\mathtt{a}'', \mathtt{w}) \wedge \texttt{AuthorHasWord}(\mathtt{a}'', \mathtt{w}') \Rightarrow \texttt{SameAuthor}(\mathtt{a}, \mathtt{a}')$. (If the names of authors $\mathtt{a}$ and $\mathtt{a}'$ respectively contain words $\mathtt{w}$ and $\mathtt{w}'$, which appear together in the name of author $\mathtt{a}''$, then $\mathtt{a}$ and $\mathtt{a}'$ are likely to refer to the same author.)

7.5 Related Work

Relational association rule mining systems (e.g., De Raedt & Dehaspe, 1997) differ from LSM in that they learn clauses without first learning motifs and are not as robust to noise (since they do not involve statistical models).

Random walks and hitting times have been successfully applied to a variety of applications, e.g., social network analysis [57], word dependency estimation [100], collaborative filtering [11], image segmentation [38], search engine query expansion [65] and paraphrase learning [44].

7.6 Conclusion

In this chapter, we presented LSM, the first MLN structure learner that is able to learn long clauses. LSM tractably learns long clauses by using random walks to find motifs of densely connected objects in data, and restricting its search for clauses to within the motifs. Our empirical comparisons with three state-of-the-art systems on three datasets demonstrate the effectiveness of LSM.

Chapter 8

CONCLUSION

Markov logic networks (MLNs) are a powerful representation that combines first-order logic and probability. MLNs attach weights to first-order clauses and view these as templates for features of Markov networks. Central to MLNs is the task of inducing their *structure*, i.e., learning the first-order clauses in them and their weights. In this thesis, we proposed several solutions to this problem.

8.1 Contributions of this Thesis

The contributions of this thesis can be summarized as follows.

- We began by combining ideas from inductive logic programming (ILP) and feature induction in Markov networks in our MSL system. MSL uses a generate-and-test approach of systematically creating candidate clauses and selecting those that directly optimize a likelihood measure (weighted pseudo-loglikelihood (WPLL)). MSL explores a large space of candidate clauses, each of which requires two computationally expensive steps: computing the clause's number of true groundings in a database and numerical optimization to find its optimal weight. We presented techniques to make the steps tractable. In our empirical evaluations, MSL outperformed the previous approach [86] of using an off-the-shelf ILP system (CLAUDIEN [18]) to first induce all first-order clauses according to some coverage and accuracy criteria, and then find their optimal WPLL weights. In addition, we also emprically showed that MSL outperformed purely ILP, purely probabilistic and purely knowledge-based approaches.

- We motivated the importance of statistical predicate invention (SPI), i.e., the discovery of new concepts, properties and relations in structured data. As a first step towards SPI, we presented the MRC system for discovering latent MLN structure. MRC is based on second-order Markov logic, in which predicates as well as arguments can be variables, and the domain

of discourse is not fully known in advance. MRC jointly clusters predicate and constant symbols, with each cluster corresponding to an invented predicate. MRC also finds multiple clusterings of the symbols, rather than just one. Our experiments showed that MRC performed better than a state-of-the-art SPI system and MSL on four relational datasets.

- We applied SPI to the long-standing AI problem of extracting knowledge from text. We created the SNE system to extract simple semantic networks from Web text in an unsupervised, domain-independent manner. SNE simultaneously clusters phrases into high-level concepts and relations, and discovers the interactions among them in the form of a semantic network. We introduced several techniques to scale SNE up to the Web. In our experiments, SNE was able to extract meaningful semantic networks from a large Web corpus and outperformed three other systems.

- We incorporated the discovery of latent MLN structure into the learning of MLN clauses in the LHL system. LHL uses data to guide its construction of candidate clauses, so as to avoid generating many candidate clauses with poor empirical adequacy and to circumvent local optima, two problems associated with top-down systems like MSL. LHL views a relational database as a hypergraph with constants as nodes and relations as hyperedges. LHL finds paths of true ground atoms in the hypergraph that are connected via their arguments. To avoid tracing the exponentially many paths in the hypergrah, LHL *lifts* the hypergraph by jointly clustering constants into high-level concepts and finding paths in the more compact hypergraph. Each path is then converted to an MLN clause. Through our experiments on three real-world datasets, we demonstrated that LHL was able to find better rules than two other state-of-the-art systems.

- MLNs use first-order formulas to define features of Markov networks, allowing potentials over very large cliques to be defined very compactly. However, this capacity is only fully exploited when long formulas are present, and previous MLN structure learners are only able to learn short clauses (4-5 literals) due to the extreme computational cost of learning. To address this problem, we presented LSM, the first MLN structure learner capable of efficiently and accurately learning long clauses. LSM is based on the observation that relational data

typically contains many patterns that are variations of the same structural motifs. By constraining the search for clauses to occur within motifs, LSM can greatly speed up the search and thereby reduce the cost of finding long clauses. LSM views a relational database as a hypergraph with constants as nodes and true ground atoms as hyperedges. LSM performs random walks on the hypergraph, identifies densely connected objects using hitting times, and groups them and their hyperedges into a motif. It then finds relational paths in the motif and converts them into clauses. Our experiments on three real-world datasets showed that LSM was 2-5 orders of magnitude faster than previous systems, while achieving the same or better predictive performance.

8.2 Future Work

This thesis has only begun the study of structure learning in MLNs. Directions for future work include the following.

An item of future work is to extend MRC and SNE to cluster predicates with different arities and argument types, so as to model the dependencies among such predicates.

Rather than inducing flat clusterings, we would like MRC and SNE to learn a hierarchy of clusterings and perform shrinkage over them. This allows information from larger top-level clusters to propagate to smaller bottom-level clusters and inform their predictions.

We would like to extend LHL and LSM from only clustering constants to clustering predicates as well. Since most domains contain many fewer predicates than constants (objects), vanilla structure learning is likely to be sufficient to model the dependencies among them. Thus, to demonstrate the usefulness of clustering predicates in LHL and LSM, we would have to apply them to rich domains with many relations, e.g., the Web.

Currently, LHL and LSM only make a single pass through the process of first creating concept clusters in a hypergraph, and then finding clauses in the hypergraph. We would like to iterate through this process such that clauses learned in one iteration could be used to find other concept clusters in the hypergraph, which can then be used to induce more clauses, and so on.

We would like to combine the top-down approach of MSL with the bottom-up approaches of LHL and LSM. For example, LSM could first learn clauses in a bottom-up manner, using data

to guide its clause construction, and then use a top-down generate-and-test approach to refine its clauses. Such a hybrid approach is in the spirit of Zelle et al. (1994) and Muggleton (1995).

In LSM, we plan to find ways to merge motifs which only differ slightly. This will help to reduce the number of motifs and further speed up LSM.

Rather than using pseudo-likelihood, which only captures short-range dependencies between an atom and its Markov blanket, we plan to tightly integrate lifted inference [97] and structure learning to allow efficient use of likelihood.

Finally, we would like to apply our algorithms to larger, more complex domains (e.g., computational biology and the Web) to test their scalability and their ability to learn complex rules.

BIBLIOGRAPHY

[1] David Aldous and Jim Fill. *Reversible Markov Chains and Random Walks on Graphs*. 2001. http://www.stat.berkeley.edu/~aldous/RWG/book.html.

[2] G. Andrew and J. Gao. Scalable training of L_1-regularized log-linear models. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, pages 33–40, Corvallis, OR, 2007. ACM Press.

[3] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, Hyderabad, India, 2007. AAAI Press.

[4] M. Banko and O. Etzioni. Strategies for lifelong knowledge extraction from the web. In *Proceedings of the Fourth International Conference on Knowledge Capture*, pages 95–102, British Columbia, Canada, 2007. ACM Press.

[5] J. Besag. Statistical analysis of non-lattice data. *The Statistician*, 24:179–195, 1975.

[6] M. Biba, S. Ferilli, and F. Esposito. Discriminative structure learning of Markov logic networks. In *Proceedings of the Eighteenth International Conference on Inductive Logic Programming*, pages 59–76, Prague, Czech Republic, 2008. Springer.

[7] M. Biba, S. Ferilli, and F. Esposito. Structure learning of Markov logic networks through iterated local search. In *Proceedings of the Eighteenth European Conference on Artificial Intelligence*, pages 361–365, Patras, Greece, 2008. IOS Press.

[8] M. Bilenko and R. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 39–48, 2003.

[9] D. M. Blei, T. Griffiths, M. I. Jordan, and J. Tenenbaum. Hierarchical topic models and the nested chinese restaurant process. In *Proceedings of the Seventeenth Conference on Neural Information Processing Systems*, pages 17–24, British Columbia, Canada, 2003.

[10] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[11] Matthew Brand. A random walks perspective on maximizing satisfaction and profit. In *Proceedings of the 8th SIAM Conference on Optimization*, Stockholm, Sweden, 2005.

[12] F. Bromberg, D. Margaritis, and Y. Honavar. Efficient Markov network structure discovery using independence tests. In *Proceedings of the Sixth SIAM International Conference on Data Mining*, Bethesda, MD, 2006.

[13] E. Charniak. *Toward a Model of Children's Story Comprehension*. PhD thesis, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Boston, MA, 1972.

[14] M. Craven and S. Slattery. Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*, 43:97–119, 2001.

[15] M. W. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to extract symbolic knowledge from the World Wide Web. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 509–516, Madison, WI, 1998. AAAI Press.

[16] J. Davis, E. Burnside, I. Dutra, D. Page, and V. S. Costa. An integrated approach to learning Bayesian networks of rules. In *Proceedings of the Sixteenth European Conference on Machine Learning*, pages 84–95, Porto, Portugal, 2005.

[17] J. Davis, I. Ong, J. Struyf, E. Burnside, D. Page, and V. S. Costa. Change of representation for statistical relational learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, Hyderabad, India, 2007.

[18] L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.

[19] M. H. DeGroot and M. J. Schervish. *Probability and Statistics*. Addison Wesley, Boston, MA, 3rd edition, 2002.

[20] L. Dehaspe. Maximum entropy modeling with clausal constraints. In *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, pages 109–125, Prague, Czech Republic, 1997. Springer.

[21] S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:380–392, 1997.

[22] W. Denham. *The detection of patterns in Alyawarra nonverbal behavior*. PhD thesis, Department of Anthropology, University of Washington, Seattle, WA, 1973.

[23] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 89–98, Washington, DC, 2003. ACM Press.

[24] P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool, 2009.

[25] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29:103–130, 1997.

[26] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, NY, 1973.

[27] M. G. Dyer. *In-Depth Understanding*. MIT Press, Cambridge, MA, 1983.

[28] G. Elidan and N. Friedman. Learning hidden variable networks: The information bottleneck approach. *Journal of Machine Learning Research*, 6:81–127, 2005.

[29] G. Elidan, N. Lotner, N. Friedman, and D. Koller. Discovering hidden variables: A structure-based approach. In *Advances in Neural Information Processing Systems 14*, pages 479–485, Cambridge, MA, 2001. MIT Press.

[30] E. Erosheva, S. Feinberg, and J. Lafferty. Mixed-membership models of scientific publications. In *Proceedings of the National Academy of Sciences of the United States of America*, 2004.

[31] O. Etzioni, M. Banko, and M. J. Cafarella. Machine reading. In *Proceedings of the 2007 AAAI Spring Symposium on Machine Reading*, Palo Alto, CA, 2007. AAAI Press.

[32] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998.

[33] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29:131–163, 1997.

[34] B. Fugledge and F. Topsoe. Jensen-Shannon divergence and Hilbert space embedding. In *IEEE International Sympsium on Information Theory*, 2004.

[35] M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA, 1987.

[36] L. Getoor and B. Taskar, editors. *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA, 2007.

[37] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, London, UK, 1996.

[38] Leo Grady and Eric L. Schwartz. Isoperimetric graph partitioning for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:469–475, 2006.

[39] T. Hasegawa, S. Sekine, and R. Grishman. Discovering relations among named entities from large corpora. In *Proceedings of the Forty-Second Annual Meeting of the Association for Computational Linguistics*, Barcelona, Spain, 2004. ACL.

[40] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[41] T. N. Huynh and R. J. Mooney. Discriminative structure and parameter learning for Markov logic networks. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning*, pages 416–423, Helsinki, Finland, 2008. ACM Press.

[42] R. Karp and M. Luby. Monte Carlo algorithms for enumeration and reliability problems. In *Proceedings of the Twenty-Fourth Symposium on Foundations of Computer Science*, pages 56–64, Tucson, AZ, 1983. IEEE Computer Society Press.

[43] C. Kemp, J. B. Tenenbaum, T. L. Griffiths, T. Yamada, and N Ueda. Learning systems of concepts with an infinite relational model. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, Boston, MA, 2006. AAAI Press.

[44] S. Kok and C. Brockett. Hitting the right paraphrases in good time. In *Proceedings of the Eleventh Conference of the North American Chapter of the Association for Computational Linguistics*, 2010.

[45] S. Kok and P. Domingos. Learning the structure of Markov logic networks. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 441–448, Bonn, Germany, 2005. ACM Press.

[46] S. Kok and P. Domingos. Statistical predicate invention. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, pages 443–440, Corvallis, OR, 2007. ACM Press.

[47] S. Kok and P. Domingos. Extracting semantic networks from text via relational clustering. In *Proceedings of the Nineteenth European Conference on Machine Learning*, pages 624–639, Antwerp, Belgium, 2008. Springer.

[48] S. Kok and P. Domingos. Learning Markov logic network structure via hypergraph lifting. In *Proceedings of the Twenty-Sixth International Conference on Machine Learning*, pages 505–512, Montreal, Canada, 2009. Omnipress.

[49] S. Kok and P. Domingos. Learning Markov logic network using structural motifs. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning*, Haifa, Israel, 2010. Omnipress.

[50] S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, J. Wang, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 2006-2010. http://alchemy.cs.washington.edu.

[51] S. Kramer. Predicate invention: A comprehensive view. Technical report, Austrian Research Institute for Artificial Intelligence, Vienna, Austria, 1995.

[52] N. Landwehr, K. Kersting, and L. De Raedt. nFOIL: Integrating naive Bayes and foil. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 795–800, Pittsburgh, PA, 2005. AAAI Press.

[53] N. Landwehr, K. Kersting, and L. De Raedt. Integrating naive Bayes and FOIL. *Journal of Machine Learning Research*, 8:481–507, 2007.

[54] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, Chichester, UK, 1994.

[55] S. Lee, V. Ganapathi, and D. Koller. Efficient structure learning of Markov networks using L_1-regularization. In *Advances in Neural Information Processing Systems 19*, Vancouver, Canada, 2007.

[56] W. G. Lehnert. *The Process of Question Answering*. Erlbaum, Hillsdale, NJ, 1978.

[57] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the Twelfth International Conference on Information and Knowledge*, pages 556–559, New Orleans, LA, 2003.

[58] B. Long, Z. M. Zhang, X. Wu, and P. S. Yu. Spectral clustering for multi-type relational data. In *Proceedings of the Twenty-Third International Conference on Machine Learning*, pages 585–592, Pittsburgh, PA, 2006. ACM Press.

[59] László Lovász. Random walks on graphs: A survey. In D. Miklós, V. T. Sós, and T. Szőnyi, editors, *Combinatorics, Paul Erdős is Eighty, Vol. 2*, pages 353–398. 1996.

[60] A. McCallum. Efficiently inducing features of conditional random fields. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 403–410, Acapulco, Mexico, 2003. Morgan Kaufmann.

[61] A. McCallum and D. Jensen. A note on the unification of information extraction and data mining using conditional-probability, relational models. In *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*, pages 79–86, Acapulco, Mexico, 2003. IJCAII.

[62] A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 169–178, 2000.

[63] A. McCallum, R. Rosenfeld, T. Mitchell, and A. Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 359–367, Madison, WI, 1998. Morgan Kaufmann.

[64] A. T. McCray. An upper level ontology for the biomedical domain. *Comparative and Functional Genomics*, 4:80–84, 2003.

[65] Qiaozhu Mei, Dengyong Zhou, and Kenneth Church. Query suggestion using hitting time. In *Proceeding of the Seventeenth ACM Conference on Information and Knowledge Management*, pages 469–478, Napa Valley, CA, 2008.

[66] L. Mihalkova and R. J. Mooney. Bottom-up learning of Markov logic network structure. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, pages 625–632, Corvallis, OR, 2007. ACM Press.

[67] T. Mitchell. Reading the web: A breakthrough goal for AI. *AI Magazine*, 26(3):12–16, 2005.

[68] R. J. Mooney. Learning for semantic parsing. In *Proceedings of the Eighth International Conference on Computational Linguistics and Intelligent Text Processing*, Mexico City, Mexico, 2007. Springer.

[69] S. Muggleton. Inverse entailment and Progol. *New Generation Computing Journal*, 13:245–286, 1995.

[70] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352, Ann Arbor, MI, 1988. Morgan Kaufmann.

[71] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the First Workshop on Algorithmic Learning Theory*, pages 368–381, Tokyo, Japan, 1990. Springer.

[72] J. Neville and D. Jensen. Leveraging relational autocorrelation with latent group models. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, New Orleans, LA, 2005.

[73] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, New York, NY, 2006.

[74] D. N. Osherson, J. Stern, O. Wilkie, M. Stob., and E. E. Smith. Default probability. *Cognitive Science*, 15:251–269, 1991.

[75] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, CA, 1988.

[76] J. Pitman. Combinatorial stochastic processes. Technical Report 621, Department of Statistics, University of California at Berkeley, Berkeley, CA, 2002.

[77] H. Poon and P. Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 458–463, Boston, MA, 2006.

[78] A. Popescul and L. H. Ungar. Cluster-based concept invention for statistical relational learning. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 665–664, Seattle, WA, 2004. ACM Press.

[79] M. R. Quillian. Semantic memory. In M. L. Minsky, editor, *Semantic Information Processing*, pages 216–270. MIT Press, Cambridge, MA, 1968.

[80] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[81] K. Rajaraman and A-H. Tan. Mining semantic networks for knowledge discovery. In *Proceedings of the Third IEEE International Conference on Data Mining*, page 633. IEEE Computer Society, 2003.

[82] P. Ravikumar, M. J. Wainwright, and J. Lafferty. High dimensional Ising model selection using L_1-regularized logistic regression. *Annals of Statistics*, 2009.

[83] J. Reisinger and M. Pasca. Latent variable models of concept-attribute attachment. In *Proceedings of the Conference of the Forty-Seventh Annual Meeting of the Association for Computational Linguistics*, 2009.

[84] B. L. Richards and R. J. Mooney. Learning relations by pathfinding. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 50–55, San Jose, CA, 1992. AAAI Press.

[85] M. Richardson and P. Domingos. Markov logic networks. Technical report, Department of Computer Science & Engineering, University of Washington, Seattle, WA, 2004.

[86] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.

[87] A. Ritter, Mausam, and O. Etzioni. A latent Dirichlet allocation method for selectional preferences. In *Proceedings of the Forty-Eighth Annual Meeting of the Association for Computational Linguistics*.

[88] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82:273–302, 1996.

[89] D. Roy, C. Kemp, V. K. Mansinghka, and J. B. Tenenbaum. Learning annotated hierarchies from relational data. In *Advances in Neural Information Processing Systems 18*, British Columbia, Canada, 2006.

[90] R. J. Rummel. Dimensionality of nations project: attributes of nations and behavior of nation dyads, 1950 -1965. ICPSR data file. 1999.

[91] Purnamrita Sarkar, Andrew W. Moore, and Amit Prakash. Fast incremental proximity search in large graphs. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

[92] R. C. Schank and C. K. Riesbeck. *Inside Computer Understanding*. Erlbaum, Hillsdale, NJ, 1981.

[93] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464, 1978.

[94] Y. Shinyama and S. Sekine. Preemptive information extraction using unrestricted relation discovery. In *Proceedings of the Seventh Conference of the North American Chapter of the Association for Computational Linguistics*, New York, New York, 2006.

[95] G. Silverstein and M. J. Pazzani. Relational clichés: Constraining constructive induction during relational learning. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 203–207, Evanston, IL, 1991. Morgan Kaufmann.

[96] P. Singla and P. Domingos. Lifted first-order belief propagation. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence*, pages 1094–1099, Chicago, IL, 2008. aaai.

[97] P. Singla and P. Domingos. Lifted first-order belief propagation. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence*, pages 1094–1099, Chicago, IL, 2008. AAAI Press.

[98] A. Srinivasan. The Aleph manual. Technical report, Computing Laboratory, Oxford University, Oxford, United Kingdom, 2000.

[99] A. Srinivasan, S. H. Muggleton, and M. Bain. Distinguishing exceptions from noise in non-monotonic learning. In *Proceedings of the Second International Workshop on Inductive Logic Programming (ILP'92)*, pages 97–107, Tokyo, Japan, 1992.

[100] K. Toutanova, C. D. Manning, and A. Y. Ng. Learning random walk models for inducing word dependency distributions. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 103–110, Alberta, Canada, 2004. ACM Press.

[101] Y. Wexler and C. Meek. Inference for multiplicative models. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, 2008.

[102] J. Wogulis and P. Langley. Improving efficiency by learning intermediate concepts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 657–662, Los Altos, CA, 1989. Morgan Kaufmann.

[103] A. P. Wolfe and D. Jensen. Playing multiple roles: discovering overlapping roles in social networks. In *Proceedings of the ICML-2004 Workshop on Statistical Relational Learning and its Connections to Other Fields*, pages 49–54, Banff, Canada, 2004. IMLS.

[104] Y. W. Wong and R. J. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the Forty-Fifth Annual Meeting of the Association for Computational Linguistics*, pages 960–967, Prague, Czech Republic, 2007. ACL.

[105] Z. Xu, V. Tresp, K. Yu, and H.-P. Kriegel. Infinite hidden relational models. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, Cambridge, MA, 2006.

[106] Z. Xu, V. Tresp, K. Yu, S. Yu, and H.-P. Kriegel. Dirichlet enhanced relational learning. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 1004–1011, Bonn, Germany, 2005. ACM Press.

[107] A. Yates and O. Etzioni. Unsupervised resolution of objects and relations on the web. In *Proceedings of the Eighth Conference of the North American Chapter of the Association for Computational Linguistics*, Rochester, NY, 2007. ACL.

[108] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Generalized belief propagation. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 689–695. MIT Press, Cambridge, MA, 2001.

[109] J. M. Zelle, R. J. Mooney, and J. B. Konvisser. Combining top-down and bottom-up techniques in inductive logic programming. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 343–351, San Mateo, California, 1994. Morgan Kaufmann.

[110] L. S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammers. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, Edinburgh, Scotland, 2005.

Appendix A

DECLARATIVE BIASES FOR CORA DOMAIN

We define "template" predicates, each of which corresponds to a clause with three literals. For example, we define $\texttt{AuthorTemplate}(\texttt{a1}, \texttt{a2}, \texttt{c1}, \texttt{c2})$ as $\texttt{AuthorTemplate}(\texttt{a1}, \texttt{a2}, \texttt{c1}, \texttt{c2}) \Leftrightarrow \neg\texttt{SameAuthor}(\texttt{a1}, \texttt{a2}) \vee \neg\texttt{AuthorOfCit}(\texttt{a1}, \texttt{c1}) \vee \neg\texttt{AuthorOfCit}(\texttt{a2}, \texttt{c2})$. (Cit is short for citation.) Similarly, we define "template" predicates $\texttt{TitleTemplate}(\texttt{t1}, \texttt{t2}, \texttt{c1}, \texttt{c2})$, $\texttt{VenueTemplate}(\texttt{v1}, \texttt{v2}, \texttt{c1}, \texttt{c2})$, and $\texttt{YearTemplate}(\texttt{y1}, \texttt{y2}, \texttt{c1}, \texttt{c2})$. A "template" predicate can be added to a clause like an ordinary predicate. When a "template" predicate appears in a clause, it is substituted by its corresponding 3-literals clause. "Template" predicates speed up the search for clauses by allowing larger search steps to be taken.

We also restrict the syntax of a clause in the following two ways.

First, when a $\texttt{CommonWordsInTitlesIsX}(\texttt{t1}, \texttt{t2})$ predicate appears in a clause, the clause must only contain $\texttt{CommonWordsInTitlesIsX}(\texttt{t1}, \texttt{t2})$ predicates and one $\texttt{SameTitle}(\texttt{t1}, \texttt{t2})$ predicate. (X can be one of $0-20\%$, $20-40\%$, $40-60\%$, $60-80\%$, and $80-100\%$.) Similar syntactic restrictions are defined for CommonWordsInAuthorsIsX/SameAuthor, and CommonWordsInVenuesIsX/SameVenue.

Second, when $\texttt{AuthorOfCit}(\texttt{a1}, \texttt{c1})$ appears in a clause, there must be exactly two AuthorOfCit predicates, one $\texttt{SameAuthor}(\texttt{a1}, \texttt{a2})$ predicate, and one $\texttt{SameCitation}(\texttt{c1}, \texttt{c2})$ predicate among the predicates in the clause. Similar syntactic restrictions are defined for TitleOfCit/SameTitle, VenueOfCit/SameVenue, and YearOfCit/SameYear.

Appendix B

MARKOV LOGIC STRUCTURE LEARNER (MSL) EXPERIMENTAL SETTINGS

When running MSL, we augmented each dataset by creating an `isX` arity-1 predicate for each constant `X` in the dataset (e.g., `isLeopard(animal)`).

The beam search version of MSL was allowed to run for 24 hours on all datasets. If MSL did not complete in 24 hours, we added the best clause in its current beam to the MLN it had found at that point, and relearned the MLN weights (by optimizing weighted pseudo-log-likelihood).

Unless stated otherwise, we used the default parameters of the Alchemy package. For all datasets, we set $minWt$ to 0, and used the $startFromEmptyMLN$ parameter. For the Animals dataset, we did not sample atoms and clauses. The length penalty was set to 0.0001. For the UML and Kinship datasets, we had to aggressively sample the number of atoms so that MSL could find some rules within 24 hours. We kept all the true atoms, and sampled the same number of false atoms as true ones. We set the length penalty to 0.001, and the $maxNumPredicates$ parameter to 4. For the Nations dataset, we did not sample atoms. The parameters $maxVars$ and $maxNumPredicates$ were both set to 10, and the length penalty was set to 0.01. All parameters were set using preliminary experiments.

To evaluate the test atoms in each fold of a dataset, we ran MC-SAT for 24 hours or 10,000,000 iterations (whichever condition occurred earlier).

Appendix C

DERIVATION OF SNE'S LOG-POSTERIOR

We show how to derive the log-posterior of SNE. Recall that an MLN, together with a set of constants, defines the probability distribution over possible worlds x as

$$P(X=x) = \frac{\exp\left(\sum_{i\in F}\sum_{j\in G_i} w_i g_j(x)\right)}{Z} \tag{C.1}$$

where Z is the partition function, F is the set of all first-order formulas in the MLN, G_i and w_i are respectively the set of groundings and weight of the ith first-order formula, and $g_j(x) = 1$ if the jth ground formula is true and $g_j(x) = 0$ otherwise (Equation 2.3).

As stated in Chapter 5, we maximize the posterior probability $P(\Gamma|R) \propto P(\Gamma, R) = P(\Gamma)P(R|\Gamma)$ where Γ is an assignment of symbols to clusters, and R is a vector of truth assignments to the observable ground atoms $r(x, y)$. The posterior probability is defined by two MLNs — one for the prior $P(\Gamma)$ component, and another for the likelihood $P(R|\Gamma)$ component.

We first derive the likelihood $P(R|\Gamma)$ component that is defined by an MLN containing the atom prediction rule. Using Equation 2.3, we get

$$P(R|\Gamma) = \frac{\exp\left(\sum_{i\in F}\sum_{j\in G_i} w_i g_j(R,\Gamma)\right)}{Z} \tag{C.2}$$

where F is the set of instances of the atom prediction rule (one instance per cluster combination with a separate weight), and G_i and w_i are respectively the set of groundings and weight of the ith instance of the atom prediction rule. Note that each grounding of an instance of the atom prediction rule contains exactly one ground atom $r(x, y)$ in its consequent. (An *antecedent* and *consequent* respectively appear on the left and right of the implication symbol $\Rightarrow$.) Since each symbol can belong to exactly one cluster, there is exactly one grounded rule containing ground atom $r(x, y)$ whose antecedent is true. All other grounded rules containing the same $r(x, y)$ atom have false antecedents, and are trivially true in all worlds. Such rules cancel themselves out in the numerator

and denominator of Equation C.2, and are ignored. Since now each ground atom $r(x, y)$ appears in the consequent of exactly one grounded rule, its truth value does not affect the truth value of other grounded rules. Therefore, the grounded rules are independent given Γ, and consequently, the $r(x, y)$ atoms are also independent given Γ. We can rewrite Equation C.2 as

$$\begin{aligned} P(R|\Gamma) &= \prod_{i \in F} \prod_{j \in H_i} \frac{\exp(w_i h_j)}{\exp(w_i h_j^0) + \exp(w_i h_j^1)} \\ &= \prod_{i \in F} \prod_{j \in H_i} \frac{\exp(w_i h_j)}{1 + \exp(w_i)} \\ &= \prod_{i \in F} \left(\frac{\exp(w_i)}{1 + \exp(w_i)} \right)^{t_i} \left(\frac{1}{1 + \exp(w_i)} \right)^{f_i} \end{aligned} \tag{C.3}$$

where H_i is the set of groundings of the ith instance of the atom prediction rule such that the antecedent of each grounding is true; h_j is the truth value of the jth grounding of the ith instance of the atom prediction rule; h_j^0 and h_j^1 are respectively the truth values of the jth grounding when the $r(x, y)$ ground atom in its consequent is set to false and true; and f_i and t_i are respectively the number of false and true $r(x, y)$ atoms that are consequents in groundings of the ith instance of the atom prediction rule. By differentiating Equation C.3 with respect to w_i, setting the derivative to 0, and solving for w_i, we find that the equation is maximized when $w_i = log(t_i/f_i)$.

Substituting $w_i = log(t_i/f_i)$ into Equation C.3 and taking logs, we obtain

$$\log P(R|\Gamma) = \sum_{i \in F} \left[t_i \log \left(\frac{t_i}{t_i + f_i} \right) + f_i \log \left(\frac{f_i}{t_i + f_i} \right) \right]. \tag{C.4}$$

Adding smoothing parameters α and β, we get

$$\log P(R|\Gamma) = \sum_{i \in F} \left[t_i \log \left(\frac{t_i + \alpha}{t_i + f_i + \alpha + \beta} \right) + f_i \log \left(\frac{f_i + \beta}{t_i + f_i + \alpha + \beta} \right) \right]. \tag{C.5}$$

We now derive the prior $P(\Gamma)$ component defined by an MLN containing three rules. The first rule is a hard rule (i.e., with infinite positive weight) stating that each symbol belongs to exactly one cluster. The second rule imposes an exponential prior on the number of cluster combinations, and it has fixed negative weight $-\lambda$. The third true encodes the belief that most symbols tend to be in different clusters, and it has fixed positive weight μ.

Using Equation 2.3, we get

$$\begin{aligned} P(\Gamma) &= \frac{\exp(\infty \cdot n_1 - \lambda n_2 + \mu n_3)}{Z} \\ &= \frac{\exp(\infty \cdot n_m + \infty \cdot (n_1 - n_m) - \lambda n_2 + \mu n_3)}{Z} \end{aligned} \tag{C.6}$$

where n_1, n_2 and n_3 are respectively the number of true groundings of the first, second and third rules; and $n_m = 1$ if the mth grounding of the first rule is true, and $n_m = 0$ otherwise. Consider the case where Γ violates the first rule. Specifically, suppose that the mth grounding of the first rule is violated. Then we get $n_m = 0$, $Z = \infty$ and $P(\Gamma) = 0$. In our search algorithm, we ensure that each symbol belongs to exactly one cluster. Thus we can remove the $\exp(\infty \cdot n_1)$ term by dividing both numerator and denominator by it. We can rewrite Equation C.6 as

$$P(\Gamma) = \frac{\exp(-\lambda n_2 + \mu n_3)}{Z'}. \tag{C.7}$$

Since λ and μ are fixed, Z' is a constant. Taking logs, we get

$$\begin{aligned} \log P(\Gamma) &= -\lambda n_2 + \mu n_3 - \log(Z') \\ &= -\lambda m_{cc} + \mu 2d - \log(Z') \end{aligned} \tag{C.8}$$

where m_{cc} is the number of cluster combinations containing true ground atoms, and d is the number of symbol pairs that are in different clusters (the 2 is due to the symmetry in the antecedent of the third rule).

Combining the likelihood and prior components, we get

$$\begin{aligned} \log P(\Gamma|R) &\propto \sum_{i \in F} \left[t_i \log \left(\frac{t_i + \alpha}{t_i + f_i + \alpha + \beta} \right) + f_i \log \left(\frac{f_i + \beta}{t_i + f_i + \alpha + \beta} \right) \right] - \lambda m_{cc} + \mu 2d - log(Z') \\ &= \sum_{i \in F} \left[t_i \log \left(\frac{t_i + \alpha}{t_i + f_i + \alpha + \beta} \right) + f_i \log \left(\frac{f_i + \beta}{t_i + f_i + \alpha + \beta} \right) \right] - \lambda m_{cc} + \mu' d + \mathcal{C} \end{aligned}$$

where $\mu' = 2\mu$ and $\mathcal{C}$ is a constant.

Appendix D

DERIVATION OF LIFTGRAPH'S LOG-POSTERIOR

In the MLN defining the prior component of the posterior probability, there are two rules. The first rule has infinite weight, and it states that each symbol belongs to exactly one cluster. The second rule has negative weight $-\infty < -\lambda < 0$, and it penalizes the number of cluster combinations. From that MLN, we get

$$P(\{\Gamma\}) = \frac{\exp\left(\infty \cdot n_{\{\Gamma\}} - \lambda m_{\{\Gamma\}}\right)}{Z} = \frac{\exp\left(\infty \cdot n_{\{\Gamma\}} - \lambda m_{\{\Gamma\}}\right)}{\sum_{\{\Gamma\}'} \exp\left(\infty \cdot n_{\{\Gamma\}'} - \lambda m_{\{\Gamma\}'}\right)} \tag{D.1}$$

where Z is the partition function; $n_{\{\Gamma\}}$ and $m_{\{\Gamma\}}$ are respectively the number of true groundings of the first and second rules for cluster assignment $\{\Gamma\}$.

We first consider the case where the first rule is violated in $\{\Gamma\}$, i.e., there is a symbol that does not belong to exactly one cluster. Note that there is a cluster assignment in which the first rule is not violated, specifically, the one where each symbol is in its own cluster. Let this cluster assignment be $\{\Gamma\}^u$. Rewriting Equation D.1, we get

$$P(\{\Gamma\}) = \frac{\exp\left(-\lambda m_{\{\Gamma\}}\right)}{\exp\left(\infty \cdot (n_{\{\Gamma\}^u} - n_{\{\Gamma\}}) - \lambda m_{\{\Gamma\}^u}\right) + \sum_{\{\Gamma\}' \setminus \{\Gamma\}^u} \exp\left(\infty \cdot (n_{\{\Gamma\}'} - n_{\{\Gamma\}}) - \lambda m_{\{\Gamma\}'}\right)}. \tag{D.2}$$

Since $n_{\{\Gamma\}} < n_{\{\Gamma\}^u}$, $0 < \lambda < \infty$, and $0 \leq m_{\Gamma^u} < \infty$, $\exp\left(\infty \cdot (n_{\{\Gamma\}^u} - n_{\{\Gamma\}}) - \lambda m_{\{\Gamma\}^u}\right) = \infty$. Consequently, the denominator of Equation D.2 is ∞, and $P(\{\Gamma\}) = 0$. Thus when the first rule is violated, the posterior $P(\{\Gamma\}|D) = 0$, and $\log P(\{\Gamma\}|D) = -\infty$.

Next we consider the case where the first rule is not violated in $\{\Gamma\}$. We divide the numerator and denominator of Equation D.1 by $\exp\left(\infty \cdot n_{\{\Gamma\}}\right)$. Let $\{\Gamma\}''$ be a cluster assignment in the summation of Z. When $\{\Gamma\}''$ violates the first rule, its contribution to the summation is zero. This is because $n_{\{\Gamma\}''} < n_{\{\Gamma\}}$ and $\exp(\infty \cdot (n_{\{\Gamma\}''} - n_{\{\Gamma\}}) - \lambda m_{\{\Gamma\}''}) = 0$. When $\{\Gamma\}''$ does not violate the first rule, $n_{\{\Gamma\}''} = n_{\{\Gamma\}}$, and $\exp(\infty \cdot (n_{\{\Gamma\}''} - n_{\{\Gamma\}}) - \lambda m_{\{\Gamma\}''}) = \exp(-\lambda m_{\{\Gamma\}''})$. Consequently, we can write Equation D.1 as

$$P(\{\Gamma\}) = \frac{\exp\left(-\lambda m_{\{\Gamma\}}\right)}{\sum_{\{\Gamma\}''} \exp\left(-\lambda m_{\{\Gamma\}''}\right)} = \frac{\exp\left(-\lambda m_{\{\Gamma\}}\right)}{Z'} \tag{D.3}$$

where the summation in the denominator is over cluster assignments that do not violate the first rule.

Taking logs, we get

$$\log P(\{\Gamma\}) = -\lambda m_{\{\Gamma\}} + K \tag{D.4}$$

where $K = -\log(Z')$ is a constant.

Next we derive the likelihood component of the posterior probability. Since each symbol x_i belongs to exactly one cluster γ_i, each ground atom $r(x_1, \ldots, x_n)$ is in exactly one cluster combination $(\gamma_1, \ldots, \gamma_n)$. Let $G_{r(x_1,\ldots,x_n)}$ be a set containing groundings of the atom prediction rules and the (single) grounding of the default atom prediction rule that have ground atom $r(x_1, \ldots, x_n)$ as their consequents. (An *antecedent* and *consequent* respectively appear on the left and right of the implication symbol $\Rightarrow$.) Suppose the cluster combination $(\gamma_1, \ldots, \gamma_n)$ to which $r(x_1, \ldots, x_n)$ belongs contains at least one true ground atom. Then there is exactly one grounded atom prediction rule in $G_{r(x_1,\ldots,x_n)}$ whose antecedent is true. The antecedents of all other rules in $G_{r(x_1,\ldots,x_n)}$ are false, and the rules are trivially true. Similarly, when cluster combination $(\gamma_1, \ldots, \gamma_n)$ does not contain any true ground atom, there is exactly one grounded *default* atom prediction rule in $G_{r(x_1,\ldots,x_n)}$ whose antecedent is true, and all other rules have false antecedents and are trivially true.

From the MLN defining the likelihood component, we get

$$P(D|\{\Gamma\}) = \frac{\exp\left(\sum_{i\in F} \sum_{j\in G_i} w_i g_j(D)\right)}{Z} \tag{D.5}$$

where Z is the partition function (different from that of Equation D.1); F is a set containing all atom prediction rules and the default atom prediction rule; G_i and w_i are respectively the set of groundings and weight of the ith rule in F; and $g_j(D) = 1$ if the jth ground rule in G_i is true and $g_j(D) = 0$ otherwise.

In the numerator of Equation D.5, we sum over all grounded rules. We can rewrite the equation by iterating over ground atoms $r(x_1, \ldots, x_n)$, and summing over grounded rules that have $r(x_1, \ldots, x_n)$ as their consequents.

$$P(D|\{\Gamma\}) = \frac{\exp\left(\sum_{r(x_1,\ldots,x_n)\in D}\sum_{j\in G_{r(x_1,\ldots,x_n)}} w_j g_j(D)\right)}{Z} \tag{D.6}$$

where $G_{r(x_1,\ldots,x_n)}$ is a set containing groundings of the atom prediction rules and the single grounding of the default atom prediction rule that have ground atom $r(x_1,\ldots,x_n)$ as their consequents; and w_j is the weight of the jth rule in $G_{r(x_1,\ldots,x_n)}$,

In $G_{r(x_1,\ldots,x_n)}$, there is exactly one grounded rule whose antecedent is true. All other grounded rules have false antecedents, and are trivially true in all worlds. Such rules cancel themselves out in the numerator and denominator of Equation D.6. Hence we only need to sum over grounded rules whose antecedents are true. We can write Equation D.6 as

$$P(D|\{\Gamma\}) = \frac{\exp\left(\sum_{r\in R}\sum_{c_r\in C_r}\sum_{j\in F_{c_r}} w_{c_r} g_j(r_j(x_1,\ldots,x_n))\right)}{Z'} \tag{D.7}$$

where R is a set of predicates; C_r is a union of cluster combinations containing at least one true grounding of predicate r, and a default cluster combination containing only false groundings of r; F_{c_r} is a set of grounded rules with cluster combination c_r in their true antecedents and a grounding of r as their consequents; w_{c_r} is the weight of the atom predication rule or default atom predication rule that has c_r in its antecedent; $r_j(x_1,\ldots,x_n)$ is the ground atom appearing as the consequent of rule j; $g_j(r_j(x_1,\ldots,x_n)) = 1$ if $r_j(x_1,\ldots,x_n)$ is true; $g_j(r_j(x_1,\ldots,x_n)) = 0$ otherwise; and Z' is the partition function.

Because a ground atom $r(x_1,\ldots,x_n)$ is in exactly one cluster combination c_r, and appears in exactly one grounded rule with c_r in its the antecedent, we can factorize Z', and write Equation D.7 as

$$\begin{aligned}
P(D|\{\Gamma\}) &= \frac{\prod_{r\in R}\prod_{c_r\in C_r}\prod_{j\in F_{c_r}}\exp\left(w_{c_r} g_j(r_j(x_1,\ldots,x_n))\right)}{\prod_{r\in R}\prod_{c_r\in C_r}\prod_{j\in F_{c_r}}\sum_{r_j(x_1,\ldots,x_n)\in\{0,1\}}\exp\left(w_{c_r} g_j(r_j(x_1,\ldots,x_n))\right)} \\
&= \prod_{r\in R}\prod_{c_r\in C_r}\prod_{j\in F_{cr}}\frac{\exp\left(w_{c_r} g_j(r_j(x_1,\ldots,x_n))\right)}{\sum_{r_j(x_1,\ldots,x_n)\in\{0,1\}}\exp\left(w_{c_r} g_j(r_j(x_1,\ldots,x_n))\right)} \\
&= \prod_{r\in R}\prod_{c_r\in C_r}\prod_{j\in F_{cr}}\frac{\exp\left(w_{c_r} g_j(r_j(x_1,\ldots,x_n))\right)}{1+\exp(w_{c_r})} \\
&= \prod_{r\in R}\prod_{c_r\in C_r}\left(\frac{\exp(w_{c_r})}{1+\exp(w_{c_r})}\right)^{t_{c_r}}\left(\frac{1}{1+\exp(w_{c_r})}\right)^{f_{c_r}}
\end{aligned} \tag{D.8}$$

where t_{c_r} and f_{c_r} are respectively the number of true and false ground $r(x_1, \ldots, x_n)$ atoms in cluster combination c_r.

By differentiating Equation D.8 with respect to w_{c_r}, setting the derivative to 0, and solving for w_{c_r}, we find that the resulting equation is maximized when $w_{c_r} = \log(t_{c_r}/f_{c_r})$. Substituting $w_{c_r} = \log(t_{c_r}/f_{c_r})$ in Equations D.8, and taking logs, we get

$$\log P(D|\{\Gamma\}) = \sum_{r \in R} \sum_{c_r \in C_r} t_{c_r} \log \left(\frac{t_{c_r}}{t_{c_r} + f_{c_r}} \right) + f_{c_r} \log \left(\frac{f_{c_r}}{t_{c_r} + f_{c_r}} \right). \tag{D.9}$$

Adding smoothing parameters α_r and β_r, we get

$$\log P(D|\{\Gamma\}) = \sum_{r \in R} \sum_{c_r \in C_r} t_{c_r} \log \left(\frac{t_{c_r} + \alpha_r}{t_{c_r} + f_{c_r} + \alpha_r + \beta_r} \right) + f_{c_r} \log \left(\frac{f_{c_r} + \beta_r}{t_{c_r} + f_{c_r} + \alpha_r + \beta_r} \right).$$

(In our experiments, we set $\alpha_r + \beta_r = 10$ and $\frac{\alpha_r}{\alpha_r + \beta_r}$ to the fraction of true groundings of r in the data.) Separating the default cluster combination c'_r containing only false groundings of r from the set of cluster combinations C_r^+ containing at least one true grounding of r, we obtain

$$\log P(D|\{\Gamma\}) = \sum_{r \in R} \left[f_{c'_r} \log \left(\frac{f_{c'_r} + \beta_r}{f_{c'_r} + \alpha_r + \beta_r} \right) + \sum_{c_r \in C_r^+} t_{c_r} \log \left(\frac{t_{c_r} + \alpha_r}{t_{c_r} + f_{c_r} + \alpha_r + \beta_r} \right) + f_{c_r} \log \left(\frac{f_{c_r} + \beta_r}{t_{c_r} + f_{c_r} + \alpha_r + \beta_r} \right) \right].$$

Using the fact that $\log P(\{\Gamma\}|D) = \log P(\{\Gamma\}) + \log P(D|\{\Gamma\}) + K'$ (where K' is a constant), and substituting $\log P(\{\Gamma\})$ and $\log P(D|\{\Gamma\})$, we get

$$\log(P(\{\Gamma\}|D) = \begin{cases} -\infty \text{ if there is a symbol that is not in exactly one cluster} \\ \sum_{r \in R} \left[f_{c'_r} \log \left(\frac{f_{c'_r} + \beta_r}{f_{c'_r} + \alpha_r + \beta_r} \right) + \sum_{c_r \in C_r^+} t_{c_r} \log \left(\frac{t_{c_r} + \alpha_r}{t_{c_r} + f_{c_r} + \alpha_r + \beta_r} \right) + f_{c_r} \log \left(\frac{f_{c_r} + \beta_r}{t_{c_r} + f_{c_r} + \alpha_r + \beta_r} \right) \right] \\ \quad -\lambda m_{\{\Gamma\}} + K'' \text{ otherwise} \end{cases}$$

where $K'' = K + K'$ is a constant. (When comparing candidate cluster assignments to find the one with the best log-posterior, we can ignore K'' because it is a constant.)

Appendix E

PROOFS OF LSM'S PROPOSITIONS

We provide below the proofs of our propositions. $\overline{p}$ denotes the reverse of path p. p_s^V denotes a path from node s to a set of nodes V. We begin by proving a lemma that is needed to prove our propositions.

Lemma A *Let v, v' and s be nodes in a ground hypergraph whose nodes are all reachable from s. If $Sym_s(v, v')$, then v and v' have the same number of r-hyperedges connected to them.*

Proof. Suppose for a contradiction that v and v' respectively have n and n' r-hyperedges connected to them, and $n > n'$. Let p_s^v be a path from s to v, $r_1, \ldots, r_n$ be the r-hyperedges that are connected to v, and $V_1, \ldots, V_n$ be the sets of nodes that are connected to v by its r-hyperedges. $V_1, \ldots, V_n$ are all distinct because a ground hypergraph cannot have more than one r-hyperedge connected to a set of nodes. (An r-hyperedge corresponds to a true ground atom, and each true ground atom can only appear once in a database.) Note that $p = p_s^v r_1 V_1 \overline{r_1 V_1} \ldots r_n V_n \overline{r_n V_n}$ is a path from s to v. We cannot create a path $p_s^{v'}$ that is symmetrical to p because $p_s^{v'}$ can contain at most $n' < n$ distinct set of nodes that are connected by r-hyperedges to v'. Hence we arrive at a contradiction that v and v' are not symmetrical. □

Proposition 1 *Let v, v' and s be nodes in a ground hypergraph whose nodes are all reachable from s, and $Sym_s(v, v')$. If an r-hyperedge connects v to a node set W, then an r-hyperedge connects v' to a node set W' that is symmetrical to W.*

Proof. Suppose for a contradiction that v' is not connected by any r-hyperedge to a node set that is symmetrical to W. Let v and v' each be respectively connected by n r-hyperedges (by Lemma A) to node sets $W_1, \ldots, W_n$ and $W'_1, \ldots, W'_n$ where $n \geq 1$ and $W_1 = W$. π_i denotes a path from s to W_i to v via r, and then back to s via the reverse path, i.e., $\pi_i = p_s^{W_i} r v \overline{p_s^{W_i} r v}$. Let $\Pi_i = \{\pi_i\}$ be

the set of all such paths. Similarly $\pi'_i = p_s^{W'_i} r v' \overline{p_s^{W'_i} r v'}$, and $\Pi'_i = \{\pi'_i\}$. Let $Q = \{\pi_1 \pi_2 \dots \pi_n\}$ be the set of paths formed by concatenating $\pi_i \in \Pi_i$. Finally let $\mathcal{Q} = \{q^1 q^2 \dots q^m p_s^v\} (m = 1, \dots, \infty)$ be the set of paths formed by concatenating $q^j \in Q$, followed by a path from s to v. Since v' is symmetrical to v, there exists $Q' = \{\pi'_1 \pi'_2 \dots \pi'_n\}$ and $\mathcal{Q}' = \{q'^1 q'^2 \dots q'^m p_s^{v'}\}$ where $q'^j \in Q'$ such that $\mathcal{Q}'$ is symmetrical to $\mathcal{Q}$. Observe that the $p_s^{W_1}$ prefix of each path in $\mathcal{Q}$ corresponds to the $p_s^{W'_1}$ prefix of each path in $\mathcal{Q}'$. Since W_1 and W'_1 are not symmetrical, there is a path in $\mathcal{Q}$ that cannot be bijectively mapped to $\mathcal{Q}'$ (or vice versa). Hence v and v' are not symmetrical, which contradicts the assumption that they are. □

Proposition 2 *The maximum value of $L_{W,C}(X)$ is attained at $W = W_0$ and $C = C_0$ where C_0 is the set of all possible conjunctions of positive ground literals that are true in X, and W_0 is the set containing the globally optimal weights of the conjunctions.*

Proof. Suppose for a contradiction $L_{W_1,C_1}(X) > L_{W_0,C_0}(X)$. First consider $W_1 \neq W_0, C_1 = C_0$. This case is not possible because by definition W_0 are the optimal weights for C_0. Next consider $C_1 \neq C_0$. For each conjunction in C_1, add all its groundings to a new set C_2. Each ground conjunction in C_2 inherits the weight of the conjunction from which it is formed. (If C_1 only contains ground conjunctions, then $C_1 = C_2$.) If C_2 contains fewer conjunctions than C_0, add these missing ground conjunctions to C_2 and give them zero weights. (W_2, C_2) thus created is equivalent to (W_1, C_1), and hence $L_{W_2,C_2}(X) > L_{W_0,C_0}(X)$. Since $C_2 = C_0$. we contradict the assumption that W_0 contains optimal weights. □

VITA

Stanley Kok was born and bred in Singapore. In 1995, he won a scholarship from the National Computer Board of Singapore (now Infocomm Development Authority (IDA)), and gleefully packed his bags to begin his undergraduate education at Brown University in Providence, Rhode Island. After spending four wonderful years there (and experiencing enough snow to last him his lifetime), he graduated with honors with a Combined Bachelor of Science (Computer Science) and Bachelor of Arts (Economics) degree. Upon returning to Singapore, he worked as an IT consultant at IDA for several years. Realizing that this first love lay in research, he began his graduate studies in the Computer Science and Engineering department at the University of Washington in Seattle. He received his M.S. in Computer Science in 2005, and a Ph.D. in the spring of 2010. Stanley's research interests are in machine learning, artificial intelligence, and their applications.