Learning and Inference in Collective Knowledge Bases

Matthew Richardson

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

Doctor of Philosophy

University of Washington

2004

Program Authorized to Offer Degree: Department of Computer Science and Engineering

University of Washington

Graduate School

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

Matthew Richardson

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 Supervisory Committee:

_____

Pedro Domingos

Reading Committee:

_____

Pedro Domingos

_____

Oren Etzioni

_____

Daniel 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 Bell and Howell Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, 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

Learning and Inference in Collective Knowledge Bases

Matthew Richardson

Chair of Supervisory Committee:
Professor Pedro Domingos
Computer Science and Engineering

Truly intelligent action requires large quantities of knowledge. Acquiring this knowledge has long been the major bottleneck preventing the rapid spread of AI systems. Hand-building comprehensive knowledge bases is slow and costly. Machine learning can be much faster and cheaper, but is limited in the depth and breadth of knowledge it can acquire. The spread of the Internet has made possible a new solution: building large knowledge bases by mass collaboration, combining information from a multitude of sources. While such *collective knowledge bases* (CKBs) promise a breakthrough in coverage and cost-effectiveness, they can only succeed if the quality, relevance, and consistency of the knowledge is kept at acceptable levels.

This dissertation introduces an architecture for collective knowledge bases that addresses these problems. It operates in two loops of interaction, one with users and one with contributors. Knowledge from contributors is used to answer questions from users, and feedback from users is used to evaluate the knowledge from contributors. By informing contributors what knowledge was used, and what may be lacking, the CKB remains relevant to the needs of its users.

Because they are developed collectively, CKBs must deal with knowledge that is inconsistent and noisy. To be of practical use, they must also be able to handle complex domains involving objects, relations, etc. Thus, this dissertation introduces *Markov logic networks* (MLNs), a knowledge representation that combines probability with the full power of first-order logic. MLNs are robust to inconsistent knowledge because they view logical statements as soft constraints as opposed to hard ones: when a world violates a formula in the KB it is less probable, but not impossible. This

dissertation introduces MLNs and develops algorithms for inference and learning in them.

The dissertation also provides an approach to combining weaker knowledge from multiple users (specifically, knowledge about which variables in a domain depend on which others), and an approach to determining the quality of contributors when insufficient data from them is available (using a web of trust among them). The utility of CKBs and the associated methods has been demonstrated with experiments in real world domains, including a public bibliography server (www.bibserv.org), a CKB about a university domain, and a CKB for printer troubleshooting.

TABLE OF CONTENTS

List of Figures **iv**

List of Tables **vii**

Chapter 1: Introduction **1**

1.1 Challenges in Building Collective Knowledge Bases 2

1.2 Knowledge Representation 3

1.3 Is Collective Construction of Knowledge Bases Feasible? 5

1.4 Overview of this Dissertation 6

Chapter 2: Background **8**

2.1 Propositional Logic 8

2.2 First-Order Logic 8

2.3 Bayesian Networks 11

2.4 Markov Networks 14

Chapter 3: Related Work **16**

3.1 Mass Collaboration 16

3.2 Probabilistic First-Order Models 20

3.3 Knowledge-Based Model Construction 22

3.4 Stochastic Logic Programs 26

3.5 Probabilistic Relational Models 28

3.6 Recent Developments in Statistical Relational Learning 32

Chapter 4: An Architecture for Collective Knowledge Bases **34**

4.1 Representation 34

4.2 Architecture 35
4.3 Algorithms 38
4.4 Experimental Evaluation 43
4.5 Limitations of KBMC 47

Chapter 5: Markov Logic Networks 49

5.1 Markov Logic 49
5.2 Relation to First-Order Logic 54
5.3 Markov Logic Subsumes SRL Approaches 55
5.4 Markov Logic Handles Key SRL Tasks 58
5.5 Implementation 60
5.6 Experiments 68
5.7 Summary 74

Chapter 6: Collective Determination of Dependency Structure 78

6.1 Approach 78
6.2 Experiments 85

Chapter 7: Using Trust Propagation to Weight Contributions 91

7.1 Model 91
7.2 Path Algebra Interpretation 93
7.3 Probabilistic Interpretation 98
7.4 Similarity of Probabilistic and Path Interpretations 100
7.5 Experiments 101
7.6 Related Work 109

Chapter 8: Conclusion 111

8.1 Contributions of this Thesis 111
8.2 Future Work 113

End Notes **117**

Bibliography **120**

Appendix A: Addendum to Chapter 4 Experimental Section **134**

A.1 Email Sent to Volunteers 134

A.2 Directions Given to Volunteers 135

Appendix B: Addendum to Chapter 5 Experimental Section **139**

B.1 Directions Given to the Volunteers. 140

B.2 Knowledge Base Produced by Volunteers 143

B.3 Algorithm Parameter Settings Used in the Experiments 148

Appendix C: Addendum to Chapter 6 Experimental Section **152**

Appendix D: Proof of Theorem 7.2.1 **171**

LIST OF FIGURES

1.1 There are many levels of precision at which we would like to be able to accept knowledge from contributors. 5

2.1 Example Bayesian network. 12

3.1 Example Bayesian network formed for a query on the knowledge base shown in table 3.1. 24

3.2 When there are multiple sources of evidence, they are combined using a node which performs a function such as noisy OR. Above is an example Bayesian network for the query "$\texttt{eats\_well}(\texttt{mary})$"?. 24

3.3 Structure for the clause $\texttt{a}(\texttt{x}) \leftarrow \texttt{b}(\texttt{X})$ remains the same regardless of query. 25

3.4 (a) Example SLP and (b) Associated SLD-tree for s(X). 27

3.5 Example PRM for a university domain (reproduced with the first author's permission from Getoor et al.[49]). 29

4.1 Input-output view of a collective knowledge base. 35

4.2 Results on synthetic knowledge bases. 45

5.1 Example ground Markov network $M_{L,C}$ where L is given by the last two rows in Table 5.1, and $C = \{\texttt{Anna}, \texttt{Bob}\} = \{\texttt{A}, \texttt{B}\}$. 53

5.2 Precision and recall for $\texttt{AdvisedBy}(\texttt{x}, \texttt{y})$, with all other predicates known (AllInfo). Each line represents a model that was trained using a different technique: Monte-Carlo maximum likelihood (ML), pseudo-likelihood (PL), or pseudo-likelihood with early stopping (PL-E). The five graphs show the results on the five different test areas . From left to right, top to bottom, they are: AI, graphics, languages, systems, and theory. 72

5.3 Precision and recall for $\texttt{AdvisedBy}(\texttt{x}, \texttt{y})$, with all other predicates known (All Info case). The five graphs show the results on the five different different test areas. From left to right, top to bottom, they are: AI, graphics, languages, systems, and theory. 75

5.4 Precision and recall for $\texttt{AdvisedBy}(\texttt{x}, \texttt{y})$, with $\texttt{Professor}(\texttt{x})$ and $\texttt{Student}(\texttt{x})$ unknown (Partial Info case). The five graphs show the results on the five different different test areas. From left to right, top to bottom, they are: AI, graphics, languages, systems, and theory. 76

6.1 Learning with knowledge from multiple experts. 79

6.2 Error probability as a function of the number of experts and their noise level. Note that $p_0 = 0.05$, so with no experts, the "best guess" is no edge between the pair of nodes, leading to an error probability of $2p_0 = 0.1$ regardless of noise level. 83

6.3 Experimental results for simulated experts: varying training set size (left) and varying noise level (right). 86

6.4 Experimental results in printer domain: low expertise (left) and high expertise (right). 88

7.1 Path Algebra belief merging on an example web of trust. 94

7.2 Strong and weak invariance. 95

7.3 Average precision ($\pm\sigma$) for *maximum* and *weighted average*. 104

7.4 Effect of λ on the precision when combining with *weighted average*. 105

7.5 Precision for various fractions of good people in the network, using *maximum* belief combination. 106

7.6 Effect of varying the quality of trust estimation. 107

LIST OF TABLES

2.1 Example of a first-order knowledge base. `Fr`() is short for `Friends`(), `Sm`() for `Smokes`(), and `Ca`() for `Cancer`(). 9

3.1 (a) Some Horn clauses defining a simple BLP for a health domain. (b) Example CPT for one of the clauses. 24

4.1 The CKB algorithms (using first-order Horn clauses and KBMC as the representation and reasoning technique). 41

4.2 Printer troubleshooting results. "Volunteer i" is the system using the ith volunteer's rules. "CKB" is the collective knowledge base. The accuracy of random guessing is 4.5%. 47

5.1 Example of a Markov logic network. `Fr`() is short for `Friends`(), `Sm`() for `Smokes`(), and `Ca`() for `Cancer`(). 52

5.2 Network construction for inference in MLNs. $MB(q)$ is the Markov blanket of q in $M_{L,C}$. 63

5.3 Algorithm used to generate flat attribute vectors for the propositional (Bayesian network and Naïve Bayes) learners. 73

5.4 Experimental results. CLL is the average conditional log-likelihood, and AUC is the area under the precision-recall curve. The results are an average over the five test sets. 77

6.1 Parameters of the expert model, $P(e_{ij}|s_j)$. 81

6.2 Network characteristics and results. p_0^* is the true probability of an arc between two nodes. δ_i is the reduction in K-L distance achieved when using i experts, as a fraction of the maximum possible (difference between learning with the empty network and learning with the the true one). 89

7.1 Average precision and recall for various belief combination functions, and their standard deviations. 103

ACKNOWLEDGMENTS

First of all, thanks definitely go to my advisor, Pedro Domingos, for all your amazing help. There is no way I would have accomplished this without you. You found me when I was wandering around, unsure of what I wanted to research, and had the perfect area of work for me.

Thank you also to my committee. Your comments and encouragement were great. I am also grateful to my mentor at IBM, Rakesh Agrawal, for always listening patiently and letting me explore freely.

Thank you Julian Besag, Alon Halevy, Henry Kautz, Tian Sang, Dan Suciu, David Heckerman, and Ramanathan Guha for fruitful discussions and help. Thanks Jimmy Lin for helping design BibServ's site, and for the hours on the phone talking about grad school.

Geoff Hulten, I received so much help from you that I don't know where to begin thanking you. Thanks for all the time you spent helping me with VFML, and for the VFML library itself. Also for encouraging discussions at low times over the past 7 years. Anhai Doan, thank you for all the encouraging conversations, both technical and not. Your advice was always great.

Sarina, you mean so much to me. You opened my heart to new things, and I will always remember that. Your caring advice, helpful conversations, and funny comments always lifted me up. We will never lose contact.

Thank you all my wonderful friends who stuck through my highs and lows. Jeremy Hance, you're a great friend who I know I can always count on. Jeremy Tantrum, thank you so much for the amount of time you have spent in the last couple of months discussing statistics and my thesis with me. Your jokes were always welcome! Thank you also Natalie for the support I feel from you, especially in the last days leading up to this final day. I appreciate all my friends at Intervarsity Graduate Christian Fellowship. Your support and listening ears and smiling smiles were wonderful.

Last but not least. Thanks to my family. Brian, Daniel, mom: I am so lucky to have each one of you in my life.

DEDICATION

For my Father. You are always in my heart.

Chapter 1

INTRODUCTION

Truly intelligent action requires large quantities of knowledge. Acquiring this knowledge has long been the major bottleneck preventing the rapid spread of AI systems. Two main approaches to this problem exist today. In the manual approach, exemplified by the Cyc project [82], human beings enter rules by hand into a knowledge base. This is a slow and costly process. Although the original goal was to complete Cyc in ten years, it has now been under development for twenty.[1] In the machine learning approach, exemplified by programs such as C4.5 [110], rules are automatically induced from data. Although this approach has been extremely successful in many domains, it has not led to the development of the large, diverse knowledge bases necessary for truly intelligent behavior. Typical learning programs contain only very weak assumptions about the world, and as a result the rules they learn are relatively shallow – they refer only to correlations between observable variables, and the same program applied to two different data sets from the same domain will typically produce different rules. Recognizing this problem, researchers have invested substantial effort into developing learning programs that can incorporate pre-existing knowledge, in effect combining the manual and automatic approaches (e.g., Pazzani & Kibler [101]). However, these programs have not been widely adopted, largely due to the difficulty and expense of capturing knowledge – the same bottleneck that has plagued purely manual solutions.

The rise of the Internet has made possible a third approach to the knowledge acquisition problem: building knowledge bases by mass collaboration, with thousands of volunteers contributing simultaneously. One motivation for this approach is the open-source software movement, which has shown that it is possible to develop very high quality software by accumulating contributions from thousands of volunteers [111]. This surprising outcome, exemplified by the success of the Linux operating system, is relevant to the construction of large-scale knowledge bases. If the work of a

large number of volunteers can be properly coordinated, knowledge bases as large as Cyc or larger can be built in a much shorter period of time, at a fraction of the cost. Conversely, over a period of a decade a knowledge base dwarfing any built so far can be inexpensively developed. We refer to such knowledge bases as *collective knowledge bases* (CKBs). The construction, inference, and learning of them is the subject of this dissertation.

The World-Wide Web itself can be seen as a large collective knowledge base. However, Web pages are mostly written in natural language, over which it is difficult or nearly impossible to reason. There are two complementary efforts underway which aim to surmount this problem: the Semantic Web, and information extraction.

The goal of the Semantic Web [10] is to build something like the existing Web, but in a machine-understandable format. Its main efforts to date have been in creating standards for the communication of data (XML), statements (RDF), ontologies (OWL), etc. We see the Semantic Web as complementary to our collective knowledge base effort, in that each can benefit the other. The Semantic Web can provide much of the infrastructure needed for collective knowledge bases (e.g., standard formats for knowledge). In turn, the methods described in this thesis can be used to guide and optimize the development of the Semantic Web.

Information extraction (IE), on the other hand, aims to extract machine-understandable information from the existing Web (e.g., Etzioni et al. [38]). Because this involves natural language processing, the results of IE are typically noisy and inconsistent. The representation and reasoning methods we develop in this thesis (see Chapter 5) are ideal for this type of knowledge. We thus see information extraction systems as potential contributors to a collective knowledge system.

1.1 Challenges in Building Collective Knowledge Bases

While collective knowledge bases promise large improvements in the speed and cost of knowledge base development, they also present many challenges. Helping to overcoming these is the primary purpose of this thesis. They are:

Quality. Ensuring the quality of knowledge contributed by many different sources, when little is known about most of them, is likely to be very difficult. We thus need mechanisms for automatically gauging the quality of contributions, and for making the best possible use of

knowledge of widely variable quality. This includes taking advantage of redundant or highly overlapping contributions, when they are available.

Consistency. As the knowledge base grows in size, maintaining consistency between knowledge entered by different contributors, or even by the same contributor at different times, becomes increasingly difficult. In a traditional logic-based system, a single inconsistency is in principle enough to make all inference collapse. This has been a major issue in the development of Cyc, and will be a much more serious problem in a knowledge base built by many loosely-coordinated volunteers.

Relevance. The initial Cyc philosophy of simply entering knowledge regardless of its possible uses is arguably one of the main reasons it has failed to have a significant impact so far. In a distributed setting, ensuring that the knowledge contributed is relevant – and that volunteers' effort is productive – is an even more significant problem.

Scalability. To achieve its full potential, a collective knowledge base must be able to assimilate the work of an arbitrarily large number of contributors, without the need for centralized human screening, coordination, or control becoming a bottleneck. Likewise, the computational learning and reasoning processes carried out within the knowledge base must scale well in the number of users, contributors, and contributions.

Motivation of contributors. To succeed, collective knowledge bases will depend on the unpaid work of a large number of volunteers. Motivating these volunteers is therefore essential. Following the example of open-source software, collective knowledge bases should allow user-developers to enter knowledge that is first of all relevant to solving their own problems. And, following the example of knowledge-sharing Web sites [39], collective knowledge bases should incorporate a fair mechanism for giving volunteers credit for their contributions.

1.2 Knowledge Representation

One of the primary choices in building knowledge bases is selecting an appropriate knowledge representation. The representation used by the CKB should be chosen to satisfy the following

desiderata:

Complexity. Many, if not most, real-world domains have complex structure. They contain multiple entities, relations, properties, etc., which cannot be properly modeled using simple propositional logic. In order to have broad applicability, the collective knowledge base should use a representation which can model these complex domains, such as first-order logic.

Uncertainty. Most real-world domains are full of uncertainty. A representation that does not explicitly model uncertainty will have very limited applicability. We thus desire a representation for CKBs that allows encoding, and reasoning with, information that is uncertain, or probabilistic, in nature.

Modularity. By its very nature, the knowledge contained in a collective knowledge base must be modular. It must be easy to add, remove, or update pieces of knowledge without needing to know, understand, or reorganize the entire collection.

Comprehensibility. Users will only contribute to something they understand. Either the representation itself, or some transformation of it, must be comprehensible. For example, logical statements can be translated into natural language, while neural networks are more difficult to interpret. Between the two, we would prefer the logical statements for comprehensibility.

Inference. We must be able to perform inference efficiently, even if approximately.

Ideally, the knowledge base should be able to accept many forms of knowledge, at many levels of precision (see Figure 1.1). This would make it possible to attract the most contributors. For example, some may feel comfortable giving only very general information such as which variables or predicates relate with which. Others may be able to give more precise statements of how they relate. More precise still would be a contribution giving some of the probabilistic parameters for the relation. All of these contributions provide potentially useful information. In Chapter 6, we present an algorithm for merging knowledge about the structure relating variables in a domain, based on an expert model that includes the probability that an expert reverses cause and effect, the probability

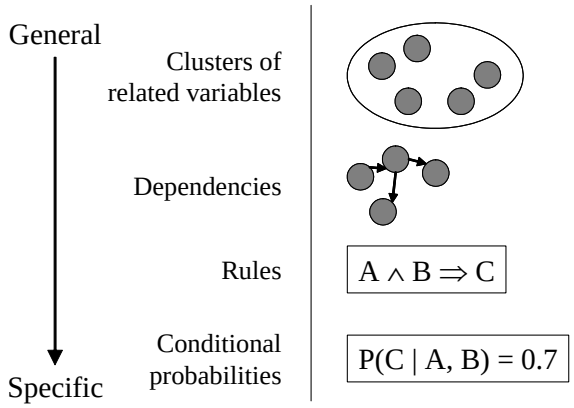


Figure 1.1: There are many levels of precision at which we would like to be able to accept knowledge from contributors.

that she adds a spurious relation, etc. We see this as a first step toward allowing collective knowledge bases to consume knowledge of many forms at many levels of detail.

1.3 Is Collective Construction of Knowledge Bases Feasible?

We may ask ourselves whether the large quantities of volunteer labor required to build large CKBs will be available. Evidence for a positive answer comes from many quarters. On the Internet today, there are hundreds of newsgroups devoted to computer troubleshooting. These, and the thousands of other newsgroups across wide-ranging topics, demonstrate the willingness of people to both ask, and answer, questions online. What is amazing about this phenomenon is the large number of users that are willing to devote their time to answering other people's questions, for little or no benefit to themselves[2]. Such seemingly altruistic behavior is beneficial in the construction of collective knowledge bases, and has been prevalent among Internet newsgroups, chat rooms, and discussion forums throughout the Internet's history.

Wikipedia (www.wikipedia.com) is another demonstration that users are very willing to contribute knowledge. Wikipedia is an "open-content encyclopedia", built, maintained, and edited completely by visitors to the site. Any visitor may add articles to the encyclopedia, or modify existing ones. Interestingly, this does not result in chaotic or garbage data. In the three years since it began, Wikipedia has amassed 261,000 articles by almost 7,000 contributors. The encyclopedia

receives approximately one to two million page requests per day, showing its usefulness as a public source of quality information.

As mentioned, the open-source movement is another motivating factor in this research. Complex open-source projects, such as the Linux operating system, Apache web server, and MySQL database engine, demonstrate that programmers are willing to donate their time and money to create something that is given away for free. This philosophy that information systems should be free, prevalent among internet users, is exactly that required for constructing a collective knowledge base.

Another encouraging sign comes from the Open Mind project [121] (www.openmind.org), which seeks to build collective knowledge bases by contributions from ordinary Internet users. The commonsense portion of Open Mind [120] currently boasts 700,000 contributions from over 14,000 people, after only four years online. 75 of these users provided 1000 or more contributions each. There seems to be no shortage of people willing to contribute to projects such as these.

This year, over 60% of all Americans are online. The result is a very diverse population of Internet users and interests. This is evidenced by the variety of chat rooms, discussion forums, and online support groups, ranging from Swedish literature to digital photography. Any such domain is a potential application for collective knowledge bases. Ideal domains for CKBs are those which have high interest and many knowledgeable people, but are too large to be easily encoded by a few. We believe there are many such domains, such as computer troubleshooting, pop culture, medicine, business, travel, cars, fashion, and food, to name a few.

1.4 Overview of this Dissertation

The next chapter reviews the basics of logic and probabilistic reasoning and inference. In Chapter 3, we examine existing work on mass collaboration, building large knowledge bases, and combining first-order logic with probability. We then present our basic architecture for collective knowledge bases in chapter 4. The architecture addresses all of the challenges given in Section 1.1. In experiments, we found that existing techniques for combining logic and probability were not sufficient for our purposes, so we created a new representation called *Markov logic*, described in detail in Chapter 5. As mentioned, we would like our knowledge base to be able to accept many forms of knowledge, so we examine how to combine dependency information from multiple experts in Chapter 6. The techniques that are introduced in Chapters 4, 5, and 6 assume that some amount of

training data is available for learning the quality of the contributors and/or contributions. We cannot always make this assumption, so we developed methods for estimating the quality of contributors without using training data. These techniques, which use a *web of trust*, are described in Chapter 7. The dissertation concludes with an overview of its contributions and directions for future work.

Chapter 2

BACKGROUND

In this chapter, we review some of the basic concepts that are used throughout the rest of this thesis. We begin with an overview of logic (propositional and first-order), including how logic programs may be learned from data. The second half of the chapter discusses probability, particularly how graphical models (Bayesian networks and Markov networks) may be used to represent and infer probability distributions, and how they are learned.

2.1 Propositional Logic

In propositional logic, terms represent *propositions*: individual statements about the domain (e.g. `cat_in_house`, `Anna_is_hungry`). Though general-purpose reasoning in propositional logic is NP-complete [19], it can be done in polynomial time if the representation is limited to propositional *Horn clauses* (see Section 2.2.2).

Propositional logic has two major limitations. First, propositions are always either known to be true, known to be false, or completely unknown. This limits our ability to model and reason in the real world, where uncertainty is prevalent. Probabilistic models, specifically *graphical models*, overcome this (see sections 2.3 and 2.4). The second limitation is that modeling a reasonably complex domain can require an exponential number of propositions. For example, to represent who is friends with whom among 20 people would require 20×20 propositions (`Friends_Anna_Bob`, `Friends_Anna_Maria`, ...). This limitation can be overcome by using first-order logic, which allows relations to be abstracted as predicates (e.g. $\texttt{Friends}(\texttt{x}, \texttt{y})$).

2.2 First-Order Logic

Logic programs and logical inference are useful tools for knowledge representation and reasoning. Here we give only a brief summary of the methods used in first-order logical deduction and induc-

Table 2.1: Example of a first-order knowledge base. $\texttt{Fr}()$ is short for $\texttt{Friends}()$, $\texttt{Sm}()$ for $\texttt{Smokes}()$, and $\texttt{Ca}()$ for $\texttt{Cancer}()$.

English	First-Order Logic	Clausal Form
Friends of friends are friends.	$\forall \texttt{x} \forall \texttt{y} \forall \texttt{z}\ \texttt{Fr}(\texttt{x},\texttt{y}) \wedge \texttt{Fr}(\texttt{y},\texttt{z}) \Rightarrow \texttt{Fr}(\texttt{x},\texttt{z})$	$\neg\texttt{Fr}(\texttt{x},\texttt{y}) \vee \neg\texttt{Fr}(\texttt{y},\texttt{z}) \vee \texttt{Fr}(\texttt{x},\texttt{z})$
Friendless people smoke.	$\forall \texttt{x}\ (\neg(\exists \texttt{y}\ \texttt{Fr}(\texttt{x},\texttt{y})) \Rightarrow \texttt{Sm}(\texttt{x}))$	$\texttt{Fr}(\texttt{x},\texttt{g}(\texttt{x})) \vee \texttt{Sm}(\texttt{x})$
Smoking causes cancer.	$\forall \texttt{x}\ \texttt{Sm}(\texttt{x}) \Rightarrow \texttt{Ca}(\texttt{x})$	$\neg\texttt{Sm}(\texttt{x}) \vee \texttt{Ca}(\texttt{x})$
If two people are friends, either both smoke or neither does.	$\forall \texttt{x} \forall \texttt{y}\ \texttt{Fr}(\texttt{x},\texttt{y}) \Rightarrow (\texttt{Sm}(\texttt{x}) \Leftrightarrow \texttt{Sm}(\texttt{y}))$	$\neg\texttt{Fr}(\texttt{x},\texttt{y}) \vee \texttt{Sm}(\texttt{x}) \vee \neg\texttt{Sm}(\texttt{y})$, $\neg\texttt{Fr}(\texttt{x},\texttt{y}) \vee \neg\texttt{Sm}(\texttt{x}) \vee \texttt{Sm}(\texttt{y})$

tion. For more on logic, logic programming, and inductive logic programming, see [45], [85], and [81], respectively

A *first-order knowledge base (KB)* is a set of sentences or *formulas* in first-order logic [45] (see Table 2.1 for an example). Formulas are built using constants, variables, functions, and predicates. Constants represent objects in the domain of interest (e.g., people: `Anna`, `Bob`, `Chris`, etc.). An *interpretation* maps constants to domain objects. Variables range over the objects in the domain. Functions (e.g., $\texttt{MotherOf}(\texttt{x})$) output an object given other objects as arguments. Predicates represent relations among objects in the domain (e.g., $\texttt{Friend}(\texttt{x},\texttt{y})$) or attributes of objects (e.g., $\texttt{Smokes}(\texttt{x})$). Predicates can have constants, variables and functions as arguments (e.g., $\texttt{Friend}(\texttt{x},\texttt{MotherOf}(\texttt{Anna}))$). A *ground predicate* is a predicate without variables. A predicate with variables can be *grounded* by replacing all variables with constants. A *world* assigns a truth value to each possible ground predicate, and is often represented compactly as a *database* or set of ground predicates by making the *closed world assumption*: if a ground predicate is not in the database, it is assumed to be false.

Formulas are recursively constructed from predicates using logical connectives and quantifiers. A formula is *satisfiable* iff there exists at least one world in which it is true. The basic inference problem in first-order logic is to determine whether a KB *entails* a formula π, i.e., if π is true

in all worlds where KB is true. This is often done by *refutation*: KB entails π iff KB $\cup \neg\pi$ is unsatisfiable. (Thus, if a KB contains a contradiction, all formulas trivially follow from it, which makes painstaking knowledge engineering a necessity.) In domains with an infinite number of constants, inference is semi-decidable; in finite domains, it is NP-complete.

2.2.1 Conjunctive Normal Form

For automated inference, it is often convenient to convert formulas to a more regular form, typically *clausal form* (also known as *conjunctive normal form (CNF)*). A KB in clausal form is a conjunction of *clauses*, a clause being a disjunction of literals (predicates or their negations). Every KB in first-order logic can be converted to clausal form using a mechanical sequence of steps. Formulas in clausal form contain no quantifiers; all variables are implicitly universally quantified. (They are also standardized apart, i.e., no variable appears in more than one clause.) Existentially quantified variables are replaced by *Skolem functions*. A Skolem function is a function of all the universally quantified variables in whose scope the corresponding existential quantifier appears. Two widely used methods for first-order inference, both using clausal form, are resolution and local satisfiability search. The latter is applied after propositionalizing the KB (i.e., forming all ground instances of first-order clauses), and proceeds by repeatedly flipping the truth values of propositions to increase the number of satisfied clauses (e.g., WalkSat [119]).

2.2.2 Horn Clauses

Because of the complexity of inference, knowledge bases are often constructed using a restricted subset of first-order logic where inference is more tractable. The most widely-used restriction is to *Horn clauses*, which are clauses containing at most one positive literal. In other words, a Horn clause is an implication with all positive antecedents, and only one (positive) literal in the consequent, or *head* (e.g., $\mathtt{a} \wedge \mathtt{b} \wedge \mathtt{c} \Rightarrow \mathtt{d}$). A program in the Prolog language is a set of Horn clauses. Prolog programs can be learned from databases by searching for Horn clauses that (approximately) hold in the data; this is studied in the field of inductive logic programming (ILP) [81].

2.2.3 Learning in First-Order Logic

Logic programs may be learned from examples. Given some background knowledge (B), and a set of examples (E), the goal is to find a set of statements (H, the hypothesis) such that B $\cup$ H entails E. Notice that this is the reverse of logical deduction, and is called *induction*, or *inductive logic programming* (ILP).

Most ILP methods can be classified into one of two categories: *top-down* or *bottom-up*. Top-down approaches, exemplified by FOIL [109], begin with an initially empty hypothesis, which is grown by adding clauses that cover positive examples. The clauses are specialized by adding antecedents such that they cover the maximum number of positive examples while misclassifying a minimum number of negative ones. This process is repeated until the hypothesis completely covers the training set.

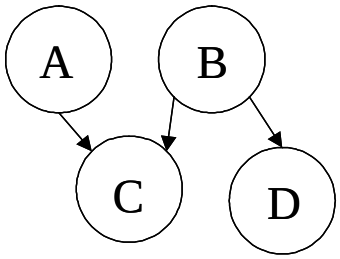
In bottom-up approaches, the hypothesis is formed by repeated generalization of the examples. This generalization is done using inductive methods such as inverse resolution [93] (used by Cigol) or mode directed inverse entailment [94] (used by Progol). Inductive methods can be thought of as "running the proof backwards", since their goal is to find the H that, if deductive methods were to be used, entails E. It is interesting to note that bottom-up ILP has the potential to create concepts not explicitly given to it. Naturally, the disadvantage of being able to create new concepts is the huge search space that results. As a result, bottom-up ILP is usually directed by multiple constraints, such as requiring that all clauses be range-restricted[1].

Earlier, we argued that the ability to explicitly represent uncertainty is necessary to model real-world domains. We now show how this can be done by using Bayesian networks.

2.3 Bayesian Networks

A Bayesian network (BN) [102] is a tuple **B**=<**G**,θ>, where **G** is a directed acyclic graph with N nodes, and θ is a set of parameters. Each node represents a random variable X_i, and edges represent probabilistic influence between nodes. A Bayesian network compactly represents the joint probability distribution P(X_1,X_2,...,X_N). The graph structure defines probabilistic independence relations: a node is probabilistically independent of its non-descendants, given its parents. As a result, the BN needs only to define $P(X_i|par(X_i))$ for each node X_i (where $par(X_i)$ is the set of parents of X_i).

Typically, this is done with a conditional probability table (CPT)[2]. If the variables are all Boolean, the probabilistic independencies encoded by the network reduce the number of parameters needed to represent the full joint probability distribution from O(2^N) to O($2^{\max_i\{|par(X_i)|\}}$). Figure 2.1 shows an example Bayesian network with four variables: A, B, C, and D.



$$P(A,B,C,D) = P(A)P(B)P(C|A,B)P(D|B)$$

Figure 2.1: Example Bayesian network.

Since a BN defines the joint probability distribution, it may be used to ask any probabilistic query about any variables in the network, given the values of any other variables. Answering these queries is the object of *probabilistic inference*, which is #P-complete in general [20]. To speed up inference on large networks, approximate inference techniques such as Markov chain Monte Carlo [53] or loopy belief propagation [95] are often used.

2.3.1 Learning with Bayesian Networks

There are two aspects of a Bayesian network that may be learned: the structure and the parameters. The distinction between learning structure and learning parameters is one that will recur throughout this thesis. When learning the parameters, the structure (**G**) is known and constant. The task then is to learn, for each node in the network, a probabilistic model of that variable given the values of its parents. The goal is typically to maximize the likelihood of the training data. If the training data is complete, this is accomplished simply by counting the co-occurrences of the values of the node with the various values of its parents[3]. When some of the data values are missing, the well-known EM algorithm [28] can be employed. EM estimates the CPTs from the known data, then uses those to estimate the missing values, then uses those to re-estimate the CPTs, and repeats until convergence.

When the structure is unknown, the task is to learn both it and the corresponding parameters. One of the most common methods for this is that of Heckerman et al. [61], which performs a search over the space of network structures, starting from an initial network which may be random, empty, or derived from prior knowledge. At each step, the algorithm generates all variations of the current network that can be obtained by adding, deleting or reversing a single arc, without creating cycles, and selects the best one using the *Bayesian Dirichlet (BD)* score

$$\begin{aligned} P(S,D) &= P(S)P(D|S) \\ &= P(S)\prod_{i=1}^{d}\prod_{j=1}^{q_i}\frac{\Gamma(n'_{ij})}{\Gamma(n'_{ij}+n_{ij})}\prod_{k=1}^{r_i}\frac{\Gamma(n'_{ijk}+n_{ijk})}{\Gamma(n'_{ijk})} \end{aligned} \tag{2.1}$$

where S is a network structure, D is a training set of n iid examples without missing values, $\Gamma()$ is the gamma function, q_i is the number of states of the Cartesian product of x_i's parents, r_i is the number of states of x_i, n_{ijk} is the number of occurrences of the kth state of x_i with the jth state of its parents, and $n_{ij} = \sum_{k=1}^{r_i} n_{ijk}$. $P(S)$ is the prior probability of the structure, which Heckerman et al. set to an exponentially decreasing function of the number of different arcs in S and the initial (prior) network. Each multinomial distribution for x_i given a state of its parents has an associated Dirichlet prior distribution with parameters n'_{ijk}, with $n'_{ij} = \sum_{k=1}^{r_i} n'_{ijk}$. These parameters can be thought of as equivalent to seeing n'_{ijk} occurrences of the corresponding states in advance of the training examples. The BD score is the result of integrating over the resulting posterior distribution for the parameters of each multinomial. The search ends, and returns the current network, when no variation achieves a higher BD score.

Note that each time the network changes, the probabilistic parameters must be updated as well. When there is no missing data, this may be done efficiently because edge additions or removals only affect the nodes that are local to the change. When there are missing values, edge changes no longer have a localized effect, in principle requiring EM to be re-run for each one. The structural EM algorithm [42] addresses this problem by filling in the missing data values with their expected value for each structural iteration. For a more thorough introduction to Bayesian networks, including how they are learned, see Heckerman [62].

2.4 Markov Networks

As with Bayesian networks, a *Markov network* [102] (also known as a *Markov random field*) is a graphical model for the joint distribution $P(X)$ of a set of variables $X = (X_1, X_2, \ldots, X_n) \in \mathcal{X}$. Unlike BNs, the graph **G** is undirected, and the parameters are specified by 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 real-valued function of the state of the corresponding clique. The joint distribution represented by a Markov network is given by

$$P(X=x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}}) \tag{2.2}$$

where $x_{\{k\}}$ is the state of the kth clique, and 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

$$P(X=x) = \frac{1}{Z} \exp\left(\sum_j w_j f_j(x) \right) \tag{2.3}$$

Inference in Markov networks is #P-complete. The most widely used method for approximate inference in Markov networks is Markov chain Monte Carlo (MCMC) [53], and in particular Gibbs sampling, which proceeds by sampling each variable in turn given its Markov blanket. (The Markov blanket of a node is the minimal set of nodes that renders it independent of the remaining network; in a Markov network, this is simply the node's neighbors in the graph.) Marginal probabilities are computed by counting over these samples; conditional probabilities are computed by running the Gibbs sampler with the conditioning variables clamped to their given values.

2.4.1 Learning with Markov Networks

As with Bayesian networks, there are two types of learning for Markov networks: structure learning and parameter learning. For parameter learning, maximum likelihood (ML) or maximum *a posteriori* (MAP) Markov network weights cannot be computed in closed form, but, because the log-likelihood is a concave function of the weights, they can be found efficiently using standard

gradient-based or quasi-Newton optimization methods [107]. Another alternative is iterative scaling [27].

Structure learning corresponds to inducing features. One method for learning features from data involves greedily constructing conjunctions of atomic features [27]. Other methods are based on maximizing likelihood in a limited domain of graph structures. For example, Chow and Liu [18] show how to use the maximum spanning tree algorithm to find the tree which best fits the data. A more general method finds the approximately best treewidth-*k* graph (a tree has treewidth of 1) [69].

Chapter 3

RELATED WORK

In this chapter, we review research related to the work in this dissertation. We first review work on mass collaboration and systems built for collective knowledge acquisition. As we will see, the choice of knowledge representation is a major issue, so we then review the state-of-the art techniques for combining first-order logic and probability.

3.1 Mass Collaboration

Gathering knowledge from experts using traditional interview methods can be time-consuming and expensive. A cheaper alternative is to solicit it over an organization's intranet, or from the relevant community of interest over the Internet, with an appropriate interface for knowledge entry. This avenue makes it possible to gather knowledge from more contributors at lower cost. This idea, using mass collaboration to collect knowledge, is not new. We give some examples of existing work in this area below.

3.1.1 Knowledge-Sharing Sites

In the introduction, we mentioned Wikipedia, an "open-content encyclopedia" which allows any visitor to add or edit articles in the collection. Wikipedia is only one of many *knowledge-sharing sites* [39], designed for the purpose of collecting, processing, and distributing knowledge. On such sites, volunteers offer advice and product ratings or help other users, typically for free. Possibly the best known knowledge-sharing site is Epinions (http://www.epinions.com). On Epinions, members submit product reviews and ratings for any of over a hundred thousand products. The Open Directory Project (www.dmoz.org) and Everything$_2$ (www.everything2.com) are two other well-known knowledge-sharing sites.

The advantages of knowledge-sharing sites are clear: By having users enter the information,

a much larger database (of product reviews, encyclopedia articles, etc) may be built, much more rapidly than otherwise possible. The disadvantages are equally clear: the resulting knowledge base will have information of highly variable quality, consistency, relevance, etc. As a result, some such sites require editors or moderators to filter out low quality information. Many sites, such as Kuro5hin (www.kuro5hin.org) distribute the moderation task itself, by letting the users themselves rate each others' postings [56]. Another approach (used by Epinions, and our own system, BibServ (www.-bibserv.org)) is to use a *web of trust*, where users rate each others' trustworthiness (see Chapter 7 for more on trust).

Perhaps the largest difference between knowledge-sharing sites and collective knowledge bases is the lack of inference and reasoning in the former. Without reasoning, such sites may only answer questions that have been directly answered in the past. In contrast, a collective knowledge base includes reasoning ability, allowing knowledge to be composed, and answers to be given that were never explicitly entered in the system.

Some primitive "inference" is performed by collaborative filtering sites and online forecasting markets. In the former, users give ratings to items (e.g. movies ratings, music tastes, etc), and the collaborative filtering system makes recommendations on items they have not yet rated (e.g. which movies to see, which CDs to buy) [112]. Similarly, Pennock et. al. studied how web-based artificial markets combine the beliefs of their users and found that they form good predictors of events [104]. Knowledge-sharing sites and collaborative filtering systems can be viewed as primitive forms of collective knowledge base. Their success is an indication of the promise of mass collaboration.

3.1.2 Open Mind

Another effort to harness the power of mass collaboration is the Open Mind Initiative (www.open-mind.org [121]). One purpose of Open Mind is to gather training sets for learning algorithms (e.g., for handwriting and speech recognition). It also has a "common sense" component [120], whose stated purpose is "to make computers smarter by making it easy and fun for people all over the world to work together to give computers the millions of pieces of ordinary knowledge that constitute 'common-sense.'"[1]. As mentioned in the introduction, this project has accumulated a large number of contributions in a very short amount of time. However, the knowledge entered into the system

is in natural language (specifically, English), making it very difficult to reason over. Also, it does not address the issues of quality, consistency, and relevance that are critical to the success of such an enterprise. As with many of the other related works, we see Open Mind as complementary to our collective knowledge base project. We believe the methods introduced in this thesis are directly applicable to the Open Mind common sense knowledge base.

Cycorp (www.cyc.com) has recently announced its intention to allow contributions to Cyc from the public (www.opencyc.org). However, its model is to have contributions screened by Cyc employees. This bottleneck prevents truly large-scale collaboration. There is also no mechanism for motivating contributors or ensuring the relevance of contributions. Another key difference between Cyc and our approach is that Cyc is an attempt to solve the extremely difficult problem of formally representing all common sense knowledge, while our goal is to build knowledge bases for well-defined, concrete domains where it should be possible to enter much useful knowledge using relatively simple representations.

3.1.3 The Semantic Web

Another attempt to gather knowledge using mass collaboration is the Semantic Web [10]. The Semantic Web's goal can be summarized as making machine-readable information available on the Web, so as to greatly broaden the spectrum of information-gathering and inference tasks that computers can carry out unaided.

Issues of quality and consistency have been largely ignored by the Semantic Web community. There has been some work [52] on computing trust on the Semantic Web using a complex, qualitative measure of trust. Otherwise, trust is often considered a Boolean attribute, in the cryptographic sense of identity verification. Our own work on trust (see Chapter 7) is equally applicable to the Semantic Web or collective knowledge bases. Similarly, our work on reasoning and representation in the presence of uncertainty, inconsistency, and low-quality of information (see Chapter 5) could be used to perform inference on the Semantic Web. Thus, the Semantic Web can be viewed as complementary to the work described in this thesis, in that each can benefit from the other. The Semantic Web can provide much of the infrastructure needed for collective knowledge bases (e.g., standard formats for knowledge). In turn, the mechanisms described in this thesis can be used to

guide and optimize the development of the Semantic Web.

3.1.4 Distributed Knowledge-Base Development

As knowledge-bases grew, it became apparent that tools were needed to support distributed development environments. Two examples of this are the Chimaera project [88] and Protege. One major difficulty and area of research in this field is the issue of maintaining a shared vocabulary. Even within a given field, different sub-communities use different terms with the same meaning, or the same term to with different meaning. Merging ontologies built by these communities is a difficult task, which may be done manually (as in McGuinness et al.[88]) or semi-automatically (see e.g., Doan et al. [30] and Dou et al. [34]).

3.1.5 The World-Wide Web

The World-Wide Web can be seen as a large collective knowledge base. Unfortunately, the majority of the information contained on the Web is in the form of natural language. However, by using the massive amounts of redundancy of this information, recent attempts at information extraction from Web pages have been quite successful [38, 79]. We believe that these techniques could complement our research on collective knowledge bases, in that information extracted from Web pages can be treated as an input from an "expert" to the knowledge base.

One form of collective knowledge on the Web that is machine-understandable is the link structure. It is impossible for any one person, or even one organization, to systematically evaluate the quality of every one of the over five billion pages on the Web today. Instead, by making the assumption that a link to a page confers some degree of confidence in the quality of that page, we may make use of the collectively-built link structure of the Web. The best-known techniques for this are PageRank [99] and HITS [75]. PageRank in particular has shown its utility, as the basic algorithm for ranking Web pages in the wildly successful search engine Google [14].

3.1.6 Model Ensembles and Belief Combination

Collective knowledge bases can be seen as combining the knowledge of many "weak" (i.e., not very accurate) experts into a "strong" knowledge base. Thus, another area of related work is that

of model ensemble methods like bagging [13], boosting [41] and stacking [127]. In essence, these methods combine many "weak" models into a "strong" one. This can produce surprisingly large improvements in accuracy.

The combination of beliefs from multiple experts has also received some attention in the statistical literature [83, 40, 46, 103]. However, this literature assumes that each expert provides a complete, self-contained probability distribution. This assumption is problematic, because human beings are notoriously poor at estimating probabilities or reasoning with them [123]. In contrast, the approach taken in this dissertation uses a division of labor between humans and machines that better reflects their respective strengths and weaknesses (cf. Jaeger [66]): humans provide statements that might be difficult for machines to discover unaided, and machines refine these statements and compute parameter estimates from data.

3.2 Probabilistic First-Order Models

Many (if not most) real-world application domains are characterized by the presence of both uncertainty and complex relational structure. For a collective knowledge base to support reasoning in such domains, it needs a representation that can handle both uncertainty and complexity. The area of statistical learning focuses on the former, and relational learning on the latter. Statistical relational learning (SRL) seeks to combine the power of both. Halpern [57] and Bacchus [7] initiated work on the problem by laying some of the necessary theoretical groundwork. Since then, a series of approaches combining features of logic and probability have been proposed, and in recent years interest in this area has grown rapidly, with a series of workshops dedicated to the topic ([48], [47], [36], [35], etc.).

Unfortunately, to our knowledge, no approach introduced so far retains the full power of both first-order logic and probabilistic graphical models. Rather, current proposals typically focus on restricted subsets of first-order logic, like Horn clauses, frame-based systems, or database query languages. This is understandable, given that logic and probability in AI are each difficult and active areas of research in their own right. Regardless, a large and growing number of SRL approaches have been proposed, including knowledge-based model construction [126, 98, 73], stochastic logic programs [90, 23], PRISM [118], probabilistic relational models [78, 43], relational Markov mod-

els [6], relational Markov networks [122], relational dependency networks [96], structural logistic regression [106], relational generation functions [22], CLP(BN) [21], and others. Knowledge-based model construction, stochastic logic programs, and probabilistic relational models are some of the earliest and best-known of these approaches. For the remainder of the chapter, we discuss these three in detail. First, we discuss the drawbacks to the simplest technique: propositionalizing the domain and applying standard probabilistic reasoning techniques.

3.2.1 Why not just propositionalize?

The majority of existing data mining and machine learning algorithms are propositional in nature (e.g. decision trees, Bayesian networks, etc.). As a result, one of the most common methods for mining relational data is simply to "flatten"[2] it into a propositional form and then apply standard mining algorithms to it. This approach has many problems. The first is that the size of a relational knowledge base can increase dramatically when flattened. Also, learning a propositional model for what is actually a first-order domain may work on one specific instance of the domain, but the model will not generalize easily to instances of different sizes. In order to generalize to different instances, the model itself needs to be relational.

Most propositional learning algorithms assume the training examples are independent of each other. This is typically a reasonable approximation, but when the data is a result of flattening a relational domain, this assumption becomes particularly incorrect. For example, given medical data about a father and son, should the training data consist of two instances (one for the father, which would include the son as an attribute, and then one for the son which would include the father as an attribute) or just one instance (describing the two of them together)? The training set that results from flattening relational data will often contain either duplicated data, or data which is ignored. Either one can lead to statistical difficulties.

Finally, using a first-order model for learning can lead to the discovery of interesting relations, or information about the relations, between entities in the model. It is very difficult to learn such relationships from a propositional model. To do so would require including all possibly interesting properties and all possibly related items as attributes in the training set, which is simply not feasible for reasonably complex domains. For these reasons, it is important to find methods for data mining

with first-order reasoning and learning abilities. In the next section, we introduce the first of these, knowledge-based model construction.

3.3 Knowledge-Based Model Construction

One way to combine probability and first-order logic is simply to augment an existing first-order (Horn clause) knowledge base with probabilistic information. This is the approach taken by *knowledge-based model construction* (KBMC) methods, as exemplified by Koller and Pfeffer [77] and by Kersting and De Raedt's *Bayesian logic programs* [72], which derive from work by Ngo and Haddaway [98] and earlier work surveyed by Wellman et al. [126]. Ng and Subrahmanian's *probabilistic logic programs* [97] are also in a similar vein. This is the approach we use in our initial CKB system, presented in Chapter 4.

The various KBMC approaches differ in their details, but the basic idea is always the same: with each clause in a knowledge base is associated a set of parameters that specify how the consequent probabilistically depends on its antecedents. In the simplest case, this is a single parameter that specifies the probability that the consequent holds given that the antecedents hold. In Kersting's Bayesian logic programs (BLPs), a complete conditional probability table (CPT) may be specified.

Table 3.1 gives an example BLP, and a CPT for one of the clauses. As can be seen, the probability that a person exercises is 0.8 if she owns a gym membership and is 0.4 otherwise. Because the clause is defined for all X, this probability distribution is the same for all people in the model. This parameter sharing facilitates both a compact representation and learning. KBMC handles any Horn clause, with relations of arbitrary arity, such as `Uses(person, gym, machine)`' (which would represent which machines a person uses to exercise).

To answer a query, KBMC extracts from the knowledge base a Bayesian network containing the relevant knowledge. Each grounded predicate that is relevant to the query appears in the Bayesian network as a node. Relevant predicates are found by using standard Prolog backward chaining techniques, except that rather than stopping when one proof tree is found, KBMC conceptually finds every possible proof tree. Further, in order to find all relevant predicates, backward chaining is done not only from the query predicate to the evidence predicates, but also from each evidence predicate to the other evidence predicates and the query predicate. The result is a set of trees which together

form a DAG (directed acyclic graph) where each node of the DAG is a ground predicate. For example, Figure 3.1 shows the Bayesian network that would result from the query "$\texttt{healthy}(\texttt{Mary})$?" given "$\texttt{eats\_well}(\texttt{Mary})$" and "$\texttt{gym\_member}(\texttt{Mary})$". The conditional probability of the node is specified by the rule's probabilistic parameters. Once the query has been converted into a Bayesian network, any standard BN inference technique may be used to answer it. Notice that the size of the Bayesian network produced by KBMC in response to a query is only proportional to the number of rules and facts relevant to the query, not the size of the whole knowledge base.

When there are multiple relevant clauses that have the same ground consequent, KBMC employs a *combination function* to compute the consequent's probability. For example, consider the second clause in Table 3.1: $\texttt{eats}(\texttt{x},\texttt{y}) \wedge \texttt{healthy\_food}(\texttt{y}) \Rightarrow \texttt{eats\_well}(\texttt{x})$. If $\texttt{eats}(\texttt{Mary},\texttt{carrots})$ and $\texttt{eats}(\texttt{Mary},\texttt{apples})$, both of which are $\texttt{healthy\_food}(.)$, then what is the probability that $\texttt{eats\_well}(\texttt{Mary})$? To answer this, an additional set of nodes are introduced in the Bayesian network, one for each clause (e.g, E1 and E2 in Figure 3.2). The node corresponding to a clause represents the proposition "All of the clause's antecedents are true," and is thus a deterministic AND function of those antecedents. For each (ground) predicate, the probability that the predicate holds is a function of the "clause" nodes (e.g., E1) that have that predicate as the consequent. For example, the predicate can be a noisy OR [102] of the clauses that have it as the consequent. Noisy OR is a probabilistic generalization of the logical OR function; it makes the assumption that the probability that one of the "causes" fails to produce the effect is independent of the success or failure of the other causes. In general, the combination function can be any model of the conditional distribution of a Boolean variable given other Boolean variables, and can be different for different predicates. Besides noisy OR, other combination functions include linear pool [40][46], and logistic regression [4].

For some combination functions (such as noisy OR with a leak node), a fact may have non-zero probability even if none of the rules for which it is a consequent apply. When using such functions, the results of inference are not complete: probabilities are only computed for facts that have at least one applicable rule. Generally, non-derivable answers vastly outnumber the derivable ones, and ignoring them greatly reduces the computational complexity of query answering. This approximation is often acceptable because the non-derivable answers will typically have low probability, rendering them less important than the derivable ones.

Table 3.1: (a) Some Horn clauses defining a simple BLP for a health domain. (b) Example CPT for one of the clauses.

$\texttt{eats\_well}(\texttt{x}) \wedge \texttt{exercises}(\texttt{x}) \Rightarrow \texttt{healthy}(\texttt{x})$
$\texttt{eats}(\texttt{x}, \texttt{y}) \wedge \texttt{healthy\_food}(\texttt{y}) \Rightarrow \texttt{eats\_well}(\texttt{x})$
$\texttt{gym\_member}(\texttt{x}) \Rightarrow \texttt{exercises}(\texttt{x})$

gym_member	$P(\texttt{exercises})$
yes	0.8
no	0.4

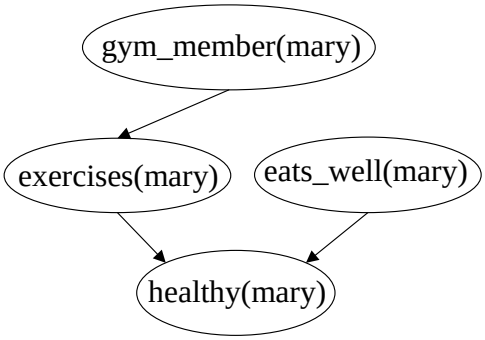


Figure 3.1: Example Bayesian network formed for a query on the knowledge base shown in table 3.1.

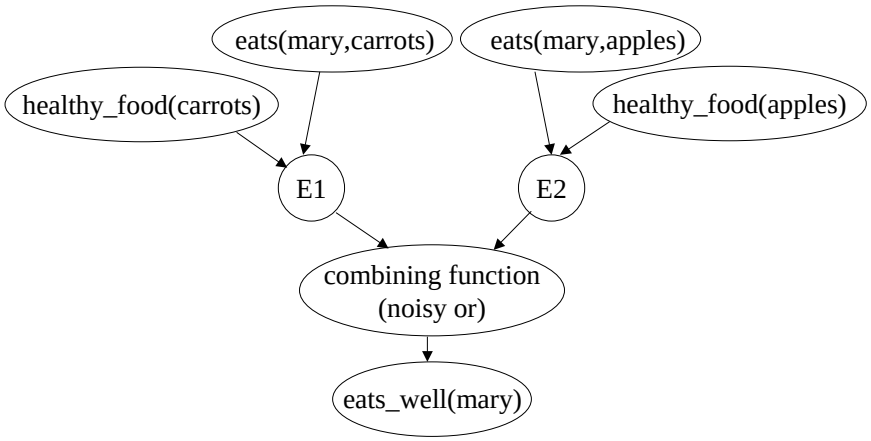


Figure 3.2: When there are multiple sources of evidence, they are combined using a node which performs a function such as noisy OR. Above is an example Bayesian network for the query "$\texttt{eats\_well}(\texttt{mary})$"?.

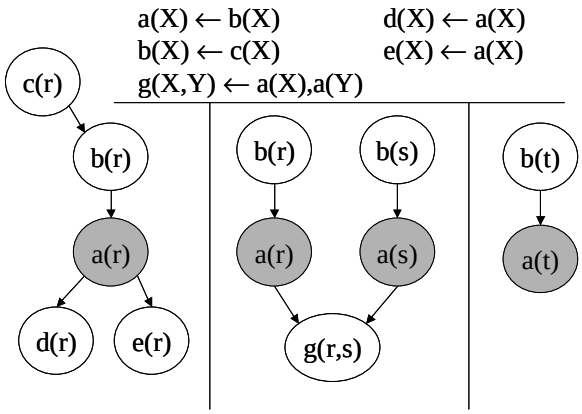


Figure 3.3: Structure for the clause $\mathtt{a(x)} \leftarrow \mathtt{b(X)}$ remains the same regardless of query.

3.3.1 Learning with KBMC

There are two parts to a KBMC knowledge base: the set of clauses, and the set of probabilistic parameters associated with them. These parts are analogous to the structure and parameters, respectively, of a Bayesian network. As with BNs, we first consider how to learn the parameters given the set of clauses.

KBMC takes as input a KB (with probabilistic parameters), evidence, and a query, and generates a Bayesian network from them. Though the network may be different for each query, the structure and CPT parameters associated with a clause will always be the same. Figure 3.3 shows this with a simple logic program and the BN that results from three different queries. Each node in the network is associated with at most one clause, and can be viewed as a separate "experiment" for it. As with Bayesian networks, the counts for each clause are simply accumulated across the training examples in order to compute its maximum likelihood parameterization. More details, along with a proof that this is correct both with complete training data and without (in which case EM is employed), can be found in Koller and Pfeffer [77] [3].

Though research on learning the "structure" (set of clauses) for KBMC has not yet been done, Kersting et al. [74] propose using methods from inductive logic programming. We believe that a heuristic search technique like those used for learning the structure of Bayesian networks, should also work.

We now discuss another method for combining probability with first-order reasoning, stochastic logic programs.

3.4 Stochastic Logic Programs

As with KBMC, a stochastic logic program (SLP) [90][92][23][25] consists of a set of first-order clauses, augmented with probabilities. However, the semantics of an SLP are very different from those of KBMC. In KBMC, inference determines a probability distribution for the value of a grounded predicate (e.g. What is the probability of $\texttt{healthy}(\texttt{Anna})$?). With an SLP, inference results in a probability distribution over the possible groundings of a given predicate (e.g. for $\texttt{healthy}(\texttt{x})$, what is the probability that $\texttt{x} = \texttt{Anna}$ vs. $\texttt{x} = \texttt{Bob}$?).

A stochastic logic program consists of a set of *stochastic clauses*, each of which has the form p:C, where C is a first-order range-restricted Horn clause, and p is a probability. Note that here we will consider only normalized, pure SLPs, that is, SLPs that have a probability associated with each clause, and in which the probabilities for any given predicate sum to 1. For a more thorough treatment, including methods for learning unnormalized and/or impure SLPs, we refer the reader to Cussens [25].

SLPs are like a first-order extension of stochastic context-free grammars (CFGs) [80]. A stochastic CFG is a CFG in which each production rule has an associated probabilistic parameter p. The parameters provide a probability distribution over production rule firings, and hence affect the distribution of terminal symbols. Similarly, in an SLP, the p parameters determine the probability distribution over first-order clause rewritings, and hence affect the distribution of ground predicates.

Inference is performed by SLD-refutation [85]. Figure 3.4 shows an example SLP and the associated SLD-tree for the goal "$\texttt{s}(\texttt{x})$?". The SLD-tree consists of all refutations of a goal via resolution. Each branch is a result of resolving with different clauses, and each path through the tree represents one complete refutation (or failure to refute). The probability of a path is the product of the probabilities of the clauses used at each branch. These probabilities are accumulated for each grounding of the query predicate:

$P(\texttt{s}(\texttt{a})) \propto 0.4 \cdot 0.3 \cdot 0.3 + 0.6 \cdot 0.2 = 0.156$	$P(\texttt{s}(\texttt{b})) \propto 0.4 \cdot 0.7 \cdot 0.7 + 0.6 \cdot 0.8 = 0.676$

To find the true probabilities, these are normalized, which results in $P(\texttt{s}(\texttt{a})) = 0.1875$, and

0.4 : s(X) p(X), p(X). 0.6 : s(X) q(X).	0.3 : p(a). 0.7 : p(b).	0.2 : q(a). 0.8 : q(b).

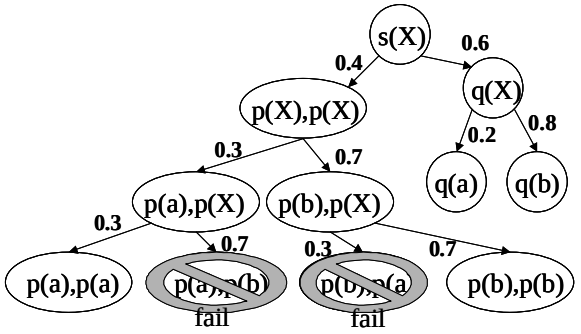


Figure 3.4: (a) Example SLP and (b) Associated SLD-tree for s(X).

$P(\mathtt{s(b)}) = 0.8125$. The normalization is necessary if any of the paths in the SLD-tree end in failure. Note that, in general, the SLD tree may be exponential in the number of nodes in the network.

SLPs can be used to represent a variety of probabilistic models, such as Bayesian networks and Markov random fields. To build these, each variable is represented by a predicate with at least one parameter, which takes value 1 or 0, indicating that the variable is true or false respectively. By computing a probability distribution over possible groundings of the predicate, SLP inference indirectly computes the probability that the variable is true and the probability that it is false. For example, to encode the predicate $\mathtt{healthy(x)}$, we would use the predicates $\mathtt{healthy(0,x)}$ and $\mathtt{healthy(1,x)}$ instead. The probability that $\mathtt{healthy(Anna)}$ is true would then be:

$$\frac{P(\mathtt{healthy(1,Anna)})}{P(\mathtt{healthy(1,Anna)}) + P(\mathtt{healthy(0,Anna)})}$$

3.4.1 Learning with Stochastic Logic Programs

As with BNs and KBMC, we distinguish between learning the structure and learning the parameters of an SLP. To learn the structure, Muggleton [91] uses standard ILP techniques, such as his Progol4.5 [94], to induce a logic program as a basis for the SLP. The clauses in the induced logic program are then augmented with uniform probabilities to create an SLP.

To learn the parameters of the SLP, Muggleton proposes a simple method which simply re-estimates the probability labels by counting the fraction of times each clause is used in a successful derivation of the goal. This simple strategy does increase the posterior probability of the model, but is sub-optimal in that it does not attempt to search for nor does it find the parameter settings which result in the highest posterior probability. It will induce the optimal parameter settings only in the case where each positive example has a unique derivation in the logic program (a scenario we believe is fairly uncommon).

Cussens [25] summarizes other techniques that have been developed for parameter estimation of SLPs. One such technique is *improved iterative scaling* [27], an algorithm that iteratively updates the parameter estimates until they converge. Riezler [115] has extended this to the case where data is incomplete, using a method analogous to EM. When the SLP is normalized, Cussens' *Failure-adjusted maximization* algorithm may be applied.

So far, we have presented two techniques for combining probability and first-order logic. Although the semantics are different, both techniques are based on a set of clauses augmented with probabilistic information. We now present the third technique, *probabilistic relational models* (PRMs), which take a different approach. Instead of adding probability to first-order logic models, PRMs conceptually add first-order elements to probabilistic models. Specifically, they add objects and relationships to Bayesian networks.

3.5 Probabilistic Relational Models

Bayesian networks have been a very successful tool for reasoning in probabilistic domains. However, they do have limitations. As a result of their propositional nature, the entirety of a domain must be known in advance, and probabilistic parameters which could be shared may end up scattered across many CPTs. For example, consider using a Bayesian network to model the processing involved in a computer chip manufacturing plant. Such plants have many machines, each of which may be affected by a variety of variables, such as the temperature of the room, the cleanliness of the air, etc. Though some of the machines may be identical, in a (propositional) Bayesian network each must be represented by its own set of nodes, with their own CPTs describing how the machine is influenced. This redundancy can have a negative effect on the efficiency of inference, as well as on

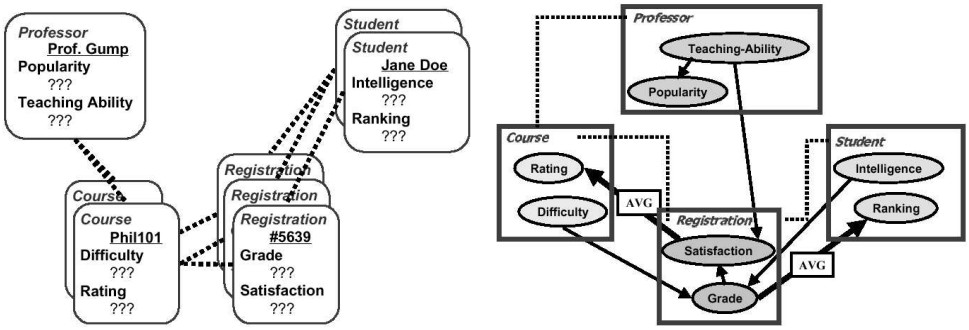


Figure 3.5: Example PRM for a university domain (reproduced with the first author's permission from Getoor et al.[49]).

the ability to learn such models due to the increased number of parameters. Furthermore, using this network to model the processing at a different plant would require adding and/or removing nodes due to machine differences between the two.

What is needed is an abstract concept of a machine, which can be instantiated multiple times (or none at all), for each type of machine in the plant. Each object would have various properties, and also relations to other objects it affects (or is affected by). Such a model was proposed by Friedman et al. [43] and is called a *probabilistic relational model* (PRM). PRMs are a form of probabilistic frame-based systems, as introduced by Koller and Pfeffer [78]. Also related is Koller et al.'s earlier work on a probabilistic description logic, P-CLASSIC [76].

A PRM consists of a set of classes $\{C_1, C_2, \ldots, C_n\}$. Each class C has a set of attributes $\mathbf{A}(C)$; each attribute A is denoted $C.A$ (For example, Person.*height*). Each class also has a set of reference slots $\mathbf{R}(C)$, where each reference slot ρ is denoted $C.\rho$, and points to an instance of the same or another class (for example, Person.*mother*). Reference slots can be composed to form a *slot chain* (for example, Person.*mother*.*father* refers to a person's maternal grandfather).

A PRM also defines the probabilistic relationship between attributes of classes. An attribute may depend on any attribute of the same class, or of a class that is reachable through some slot-chain. Figure 3.5 shows an example PRM. Reference slots are denoted by dotted lines between classes (e.g. Registration.*course* and Registration.*student*). As can be seen in the figure, PRMs allow relations as simple as "A professor's popularity depends on his teaching ability" or as complex as "A student's

grade in a class depends on the difficulty of the class and the intelligence of the student". Note that slot chains are not always one-to-one. In the example PRM, a student's ranking depends on the grade he receives in a course, but since there are multiple courses, there must be some way to combine them when determining the ranking of the student. For this, PRMs use the concept of an *aggregation function*, which has a similar role to that of combining functions in KBMC. An aggregation function combines a set of attributes into a single value. In this example, the student's ranking depends probabilistically on the average of the grades he received.

A PRM can be thought of as a template which, when given a specific domain of objects, is "compiled" into a Bayesian network. Given a PRM and a set of objects, inference is performed by constructing the corresponding Bayesian network of and applying standard inference techniques to it (e.g., belief propagation).

3.5.1 Learning with Probabilistic Relational Models

As before, learning in PRMs can be divided into two tasks: learning the structure of the model, and learning the parameters. For PRMs, the "structure" is the relational skeleton, which specifies the parent set $par(C.A)$ for all attributes A of all classes C.

When the structure is known (along with the set of all objects in the domain and the relationships between them), learning the parameters is nearly identical to learning the parameters of a standard Bayesian network. The only difference is that some parameters are *tied*, or forced to take the same value. The resulting likelihood function still decomposes into a product of terms which may be individually maximized, as in BNs, using simple counts.

Structure learning is also similar to BNs. A hill-climbing procedure searches through the space of possible structures using operations such as adding, deleting, or reversing edges in the PRM. The search is directed by a model scoring function, as with standard Bayesian networks. However, considering all possible connected variables would be intractable, so instead the search is done in stages, first searching over slot-chains of length zero, then one, etc. For more details, such as ensuring that the PRM always induces an acyclic BN, see [43].

3.5.2 Structural Uncertainty

A standard PRM has full knowledge of the relations between objects. Getoor et al. [50] relax this requirement by introducing the *probabilistic relational model II* (PRM2). A PRM2 allows two types of uncertainty over the relationships between objects: *reference uncertainty* and *existence uncertainty*.

In *reference uncertainty*, the objects in the domain are known, but the references between them are not (though the number of relations is known). For each unknown reference slot, the PRM2 induces a distribution over objects, which determines the probability that the given object is assigned to that slot. However, because the PRM2 must be able to generalize for any set of objects, it cannot directly specify a probability distribution over the objects of the training set. Instead, it indirectly selects objects according to their attributes.

For each reference slot, objects are first partitioned according to some of their attributes. Then, from the training set, the PRM2 learns a probability distribution over partitions. This distribution is just like the distribution of any other attribute, in that it is conditioned on some other set of attributes, which may be in the same class or in some other class via a slot chain (these can be thought of as the parents of the slot). Each slot has a CPT which specifies the probability distribution over partitions, given its parents. The probability mass assigned to a partition is then uniformly distributed over the objects contained within.

For example, consider the university domain given above in Figure 3.5. The relation Course.*taught-by* may be uncertain. The PRM2 defines some set of attributes, say {*department,seniority*}, by which to partition all of the professors (not shown in the figure). The probability distribution over departments is specified by a CPT which may have as parents, for instance, Course.*subject* and Course.*difficulty*. From the training set, the PRM2 may learn, for example, that the Course.*taught-by* relation for easy computer courses is more likely to be filled by new professors in computer science than by senior professors in linguistics.

If the partition attributes for each relation are provided, then a PRM2 may be learned in the same manner as a standard PRM (with only slight modifications). However, the partition attributes are not assumed to be known in advance. To find them, the hill-climbing structural search procedure is augmented with two new actions: *refine* and *abstract* which add or remove (respectively) an attribute

from the partition set for a relation (which is initially empty).

The second type of structural uncertainty introduced in PRM2s is *existence uncertainty*. Recall that, in reference uncertainty, the number of relations is known, but the relations between objects are not. In existence uncertainty, not even the number of relations is known. To handle this, Getoor et al. augment classes with an existence (E) attribute, which is *true* if the object exists and *false* if it does not. $C.E$ is treated as any other attribute, in that it may have parents, and is associated with some CPT. The result is a probabilistic model for the existence of relations between objects.

For example, consider the student domain and suppose that the registration objects are undefined. The Registration.E attribute may have, as parents, the student's major and year of study, and also the course subject. The associated CPT would likely represent that a registration object is more likely to exist if the major and year of study of the student match the department and difficulty of the course. The model (conceptually) instantiates all possible registration objects, assigning to each a probability of existence based on this.

$C.E$ is treated like any other attribute of the class, so learning a PRM2 with existence uncertainty is also a simple modification of the standard PRM algorithm.

PRMs provide a useful framework for modeling domains that contain both uncertainty and relational structure. Although PRMs can represent only a subset of first-order logic, their basis in objects and relationships is a natural fit to the way in which domains are often understood and described. As a result, they are a useful tool for modeling complex, real-world domains. PRMs have also been extended recently to dynamic domains, where the objects and relations may change over time [117].

3.6 Recent Developments in Statistical Relational Learning

In the past few years, interest in Statistical Relational Learning (SRL) has grown rapidly, resulting in many new techniques, many of which are extensions to existing propositional methods. This section gives an overview of a few of these.

Relational Markov models. In Anderson et al.'s RMMs, the states of the Markov model are labeled with parameterized predicates [6]. By using a first-order representation for the state, RMMs are, among other things, better able to use smoothing to combat data scarcity.

Relational Markov networks. RMNs [122] are a combination of Markov networks and database queries. The relational structure of the RMN is defined by *relational clique templates*, which are essentially SQL queries, and their associated potential functions. Each clique template, applied to a database, generates a set of tuples. Each tuple defines a clique in the "unrolled" ground Markov network. RMNs have been shown to be useful in a collective classification task. Because they use one parameter per state of the clique, RMNs are limited to fairly small cliques.

Structural Logistic Regression. In structural logistic regression (SLR) [106], the predictors are the output of SQL queries over the input data.

Plates. Plates are a convenient way to represent large graphical models that have repeated structure [54]. A simple example of this is dynamic Bayesian networks, for which the entire time slice is one plate. Embedding or overlapping plates allow the representation of relational structure.

Relational Dependency Networks. A relational dependency network is a dependency network in which each node's probability, conditioned on its Markov blanket, is given by a decision tree over relational attributes [96].

Chapter 4

AN ARCHITECTURE FOR COLLECTIVE KNOWLEDGE BASES

In this chapter, we present our architecture for collective knowledge bases, which addresses the five challenges outlined in the introduction. The architecture can in principle be applied with a variety of representations and inference methods; we initially used first-order Horn clauses and KBMC (see Section 3.3). We demonstrate the utility of collective knowledge bases through experiments. The chapter concludes by identifying some of the problems with the KBMC representation, leading to the next chapter in which we present a novel representation that overcomes them.

4.1 Representation

In choosing a representation, we must keep in mind the five issues from Chapter 1: complexity, uncertainty, modularity, comprehensibility, and inference. The first two suggest a method that combines first-order logic with probability. As we saw in Chapter 3, there are many such methods. One of them, SLPs, assumes that for a given consequent, only one rule can fire at a time. SLPs are thus not a natural fit when multiple rules can function simultaneously as sources of evidence for their common consequent. Another method, probabilistic relational models, lacks the modularity required for construction by many loosely-coordinated individuals. The various relational methods available today are mostly not as modular or comprehensible as, say, a collection of first-order rules. Further, most of them did not yet exist at the time we were creating this architecture.

KBMC, on the other hand, fulfills all the desiderata. Horn clauses have the key feature of high modularity: a new rule can be input without knowing what other rules are already in the knowledge base. Horn rules are also very comprehensible: rules can be read as "if-then" statements, making them natural for reading and writing. The only apparent drawback to using KBMC is its use of Horn clauses instead of full first-order logic. In their defense, however, Horn clauses *are* used in many expert system shells, and form the basis of the Prolog programming language. They

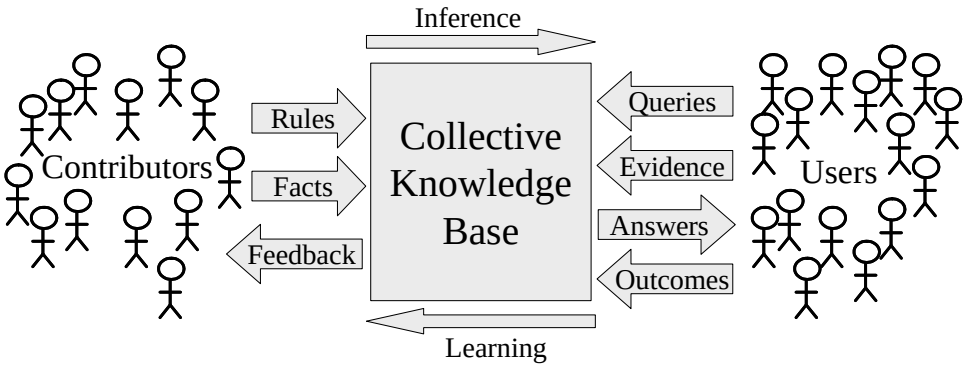


Figure 4.1: Input-output view of a collective knowledge base.

are an effective trade-off between expressiveness and tractability, and using them takes advantage of extensive previous research on making Horn-clause inference efficient. We thus present the architecture, and experiments with it, using KBMC for inference.

4.2 Architecture

Figure 4.1 shows an input-output view of the architecture. A collective knowledge base is a continuously-operating system that receives three streams of information:

Rules and facts from contributors. Note that, although our current system requires entering knowledge directly in Horn-clause form, this need not be the case in general. Allowing knowledge entry via menu-driven interfaces, ontology browsers, and restricted forms of natural language (e.g., using a particular syntax, or within specific domains) should greatly increase the number of individuals that are able to contribute. The extensive previous research on tools for knowledge base development should be useful here (see Section 3.1.4).

Queries and evidence from users. Following conventional usage, a query is a predicate with open variables (input directly, or obtained by translation from a user interface). Users also supply evidence relevant to the queries in the form of a set of facts (ground instances of predicates). These facts may be manually input by the user, or automatically captured from the outside system the query refers to. (For example, if the query is a request to diagnose a malfunctioning

artifact such as a car or a computer, information on the state and configuration of the artifact may be captured directly from it.) As in many knowledge-sharing sites, queries can also have a "utility value" attached, reflecting how much the user is willing to "pay" (in some real or virtual unit) for the answer.

Feedback on the system's replies, from users ("Outcomes"). Given the answer or answers to a query, the user takes actions, observes their outcomes, and reports the results to the knowledge base. For example, if the query is "Where on the Web can I find X?" and the answer is a URL, the user can go to that URL and report whether or not X was found there. In a fault diagnosis problem, the user attempts to fix the fault where diagnosed, and reports the result. The outcome can also be in the form of a utility rating for the answer. This rating can be objective (e.g., time saved, number of "hits" in some task) or subjective (e.g., user's satisfaction on a five point scale). The rating could either be given explicitly (e.g., ask the user directly) or implicity (e.g., assume that the problem was solved if the user does not ask again).

In return, the collective knowledge base produces two streams of information:

Answers to queries. Answers to a query consist of instantiations of the open variables for which the query predicate holds true. They are sorted by probability of correctness, in the same way that search engines sort documents by relevance. Probabilities of correctness are computed as described below.

Feedback to contributors. Contributors receive from the knowledge base feedback on the quality of their entries, in the form of accumulated (positive or negative) credit for their use in answering queries. The credit assignment computation is described below.

The collective knowledge base is thus involved in two continuous loops of interaction, one with contributors, and one with users (these two populations need not be disjoint). Contributors and users, as a result, interact via the knowledge base. This interaction is in general not one-to-one, but many-to-many: entries from many different contributors may be combined by inference to yield the

answer(s) to a query, and the feedback from a query's outcome will in return be propagated to many different contributors. Conversely, a single contribution may be used in answering many different queries, and receive feedback from all of them.

A key feature of this architecture is that collective knowledge bases are built by an intimate combination of human work and machine learning, and the division of labor between them reflects their respective strengths and weaknesses. Human beings are best at making simplified, qualitative statements about what is true in the world, and using their judgment to gauge the quality of the end results produced by computers. They are notoriously poor at estimating probabilities or reasoning with them [123]. Machines are best at handling large volumes of data, estimating probabilities, and computing with them. Another key feature is that the knowledge base is not developed in open-loop mode, with the knowledge enterers receiving no real-world feedback on the quality and correctness of their contributions. Rather, the evolving knowledge base is subjected to constant reality checks in the form of queries and their outcomes, and the resulting knowledge is therefore much more likely to be both relevant and correct.

In current knowledge-sharing sites and knowledge management systems, questions are answered from an indexed repository of past answers, or routed to the appropriate experts. Thus the only questions that can be answered automatically are those that have been asked and answered in the past. In contrast, the architecture we propose here allows chaining between rules and facts provided by different experts, and thus automatically answering potentially a very large number of questions that were not answered before. This can greatly increase the utility of the system, decrease the cost of answering questions, and increase the rewards of contributing knowledge.

The architecture answers the problems posed in the introduction:

Quality. By employing feedback and machine learning, we are able to determine which rules are of high quality, and which are not. Further, since we are tracking the utility of knowledge provided by users, they are more inclined to provide good rules.

Consistency. By using a probabilistic framework, we are able to handle inconsistent knowledge.

Relevance. Since the knowledge base is being built by users, for users, we expect the rules to be on topics that the users find relevant and interesting. The credit assignment process rewards those

contributors whose rules are used (and produce correct answers), which provides incentive to create rules that are relevant to users' needs.

Scalability. For both training and query-answering, the most expensive portion of the computation is the probabilistic inference on the Bayesian network. However, this computation depends only on the size of the network, not of the entire knowledge base. The Bayesian network is constructed out of only the relevant knowledge, which we expect (and confirm empirically in the experimental section) will typically lead to relatively small networks even for very large knowledge bases.

Motivation of contributors. By tracking the utility of rules and assigning credit to those which are used to answer queries, we provide the means for motivating contributors (e.g. listing the top ten, paying in some real or virtual currency, etc.)

In the next section, we present the CKB algorithms in more detail.

4.3 Algorithms

There are three primary algorithms necessary to implement the CKB architecture as introduced. The first, inference, uses knowledge in the CKB to answer user queries. The second, learning, uses the feedback from users to update its internal state and understanding of the domain. The third, credit assignment, distributes credit among contributors according to the utility of their knowledge. In this section we describe these three algorithms, with respect to our choice of knowledge representation: first-order Horn clauses.

4.3.1 Inference

The goal of inference is to answer a query, given some set of evidences. In our case, inference is performed using KBMC as described in Section 3.3. First, KBMC constructs a Bayesian network that contains all of the relevant knowledge necessary to answer a query, and then uses standard inference techniques on the Bayesian network to answer the query.

To understand KBMC, it helps to consider the full ground network induced by a knowledge base. Given some first-order Horn-clause KB, let $B(KB)$ be the Bayesian network that results from the

following process: create one node per ground clause, and one node per ground predicate. Let the parents of a clause node be the clause's antecedents, and the conditional probability of the node be an AND of the antecedents. Let the parents of a predicate node be the clause nodes for which it is the consequent; the conditional probability of the node may be any combination function, such as noisy OR or logistic regression.

By performing Bayesian inference on $B(KB)$, we may answer any query, given any evidence. However, to do so would be highly intractable, due to the size of the network. Instead, KBMC first aims to build the minimal BN that still represents the same conditional probability of the query given the evidence.

Let $R = \{q, E\}$ be the set containing the query and evidence nodes. For each $r \in R$, we find all possible proofs of r by backward-chaining to nodes in R. Each proof forms a tree (with the leaves being nodes in R); a path through the tree is an alternating sequence of ground predicates and ground clauses — the ground predicate being the consequent of the clause that is next in the path (toward the leaves), and the antecedent of the clause that was previous. Note that this tree will necessarily be a subgraph of $B(KB)$. The set of proof trees is converted to a graph by creating a node for any ground predicate or ground clause that appears in the proofs, and adding the arcs found in the proof trees. The resulting DAG is a subset of $B(KB)$, and is the structure of the Bayesian network on which inference is to be performed. The parameters of the Bayesian network come directly from the KB.

Our implementation of KBMC uses gprolog, an open-source Prolog interpreter, to find the proof trees via backward chaining. Since Prolog returns with an answer to a query, but not the proof of it, it was necessary to augment each predicate with an additional "history" variable which maintained a list of the clauses involved in the proof. Since each proof of the query resulted in a different assignment to this variable, we could obtain the list of all proofs by requesting all solutions to the query predicate with the history variable unbound.

4.3.2 Learning

The goal of learning is to find the clause weights that maximize the likelihood of the user feedback. Again, imagine the full ground Bayesian network, $B(KB)$. Each feedback instance consists of a

query predicate, evidence predicates, and the correct assignment to the query node. It is thus simply an assignment to some of the variables in $B(KB)$, leaving the rest unknown. The EM algorithm [28] is the most well-known method for learning such problems with missing data, and is similarly applied to learning KBMC weights. However, notice that multiple parameters in $B(KB)$ correspond to the same parameter in KB. This is an example of *parameter tying*; Koller and Pfeffer [77] showed that with only a small modification, EM can be applied to this problem. Our implementation uses this technique.

4.3.3 Credit Assignment

The goal of credit assignment is to distribute credit to contributors based on the quality of their contributed knowledge. Given feedback on the correctness of the answer (entered by the user), the utility of the corresponding query is propagated throughout its proof DAG. Credit is divided equally among the proof trees, and, within each tree, among the rules and facts that were used in it. If a rule or fact was used in multiple proof trees, it accumulates credit from all of them. Over many queries, each rule or fact accumulates credit proportional to its overall utility, and each contributor accumulates credit from all the queries his/her knowledge has helped to answer. If a rule or fact tends to lead to incorrect answers, the consequent cost (or negative utility) will be propagated to it; this will encourage contributors to enter only knowledge they believe to be of high quality.

The algorithms are summarized in Table 4.1. The function **Inference** builds the Bayesian network neccessary to answer a particular query by first retrieving the necessary structure and then assigning the associated parameters to that structure. It then uses Bayesian network inference on that network to answer the query. In our implementation, we use likelihood-weighted sampling for inference. **Learn** updates the parameters of the clauses in the KB so that the likelihood of the feedback from users is maximized. It does this by using EM: In the expectation stage, the unknown predicates are sampled, and in the maximization stage, the samples are used to maximize the KB parameters. **BuildDAG** finds all of the nodes that must be in the Bayesian network in order to answer a particular query. It does this by backward-chaining from each evidence or query node to all the other evidence or query nodes, and then combining these proof trees into a single graph. **BuildBN** takes in a graph structure and assigns to it the necessary parameters to make it a Bayesian network.

Table 4.1: The CKB algorithms (using first-order Horn clauses and KBMC as the representation and reasoning technique).

Function Inputs:

E Set of grounded predicates (evidence)
q A query predicate
F Feedback set $\{f_1, f_2, \ldots\}$, with $f_i = (\mathbf{E}, q, a)$:
an evidence set, a query, and the correct answer.
G A graph structure

Global Variables:
KB Set of horn clauses with associated probabilities (knowledge base)

Inference(q, E)
Returns the probability $P(q|E)$.
G=BuildDAG(q,E)
BN=BuildBN(G)
Return BN.Infer($P(q|E)$)

Learn(q, E)
Updates the parameters in KB to maximize the likelihood of the user feedback.
For each f_i in F:
G_i = BuildDAG($f_i(q)$, f_i(E))
Do while KB probabilities have not converged:
For each f_i in F:
BN_i = BuildBN(G_i)
Samples = BN_i.GenerateSamples()
E_i = Samples.ExpectedValues()
KB = MaximizeLikelihood(E)

AssignCredit(F)
Assigns credit (or discredit) to contributors.
For each f_i in F:
B = {f_i(E),$f_i(q)$}
For each b in B:
$Trees$ = GetProofTrees(b,B$\setminus b$)
$credit = \frac{1}{|Trees|}$
If $f_i(q) \neq f_i(a)$: $credit = -credit$
For each $Tree_j$ in Trees
For each $Clause_k$ in $Tree_j$
Assign $\frac{credit}{|Tree_j|}$ credit to contributor of $Clause_k$

BuildDAG(q, E)
Builds the graph structure needed to answer the query $P(q|E)$.
G = {}
B = {E,q}
For each b in B:
Trees = GetProofTrees(b,B\ b)
For each $tree_i$ in Trees:
For each node in $tree_i$: add node to G
For each arc in $tree_i$: add arc to G
Return G

BuildBN(G)
Takes in a graph structure and builds a BN using the parameters in the KB.
BN.structure = G
For each $node_i$ in G
If $node_i$ is a clause:
$node_i$.parameters = "AND" of parents
If $node_i$ is a predicate:
$node_i$.parameters = combination function of parents (which are ground clauses), using weights of parents as parameters
Return BN

GetProofTrees(b, B)
Returns all proof trees of some predicate b given some other predicates B.
For each clause C_i in KB:
D_i = C_i with additional history variables
H = Prolog(b, D $\cup$ B)
Construct trees from H
Return trees

Each node which represents a ground clause is assigned a CPT representing the AND function of its antecedents, and nodes which represent ground predicates are assigned a function based on the user's choice of combination function and the weights in KB. **GetProofTrees** calls Prolog to find all of the proof trees of the given predicate and evidence. **MaximizeLikelihood** finds the parameters of KB that maximize the likelihood of the expectations, E, computed in the E-step of EM. The implementation of this function is dependent on the particular combination function being used. For example, it may employ logisitic regression.

4.4 Experimental Evaluation

We performed two sets of experiments using our implementation of a collective knowledge base, which uses logistic regression as a combination function, and likelihood-weighted sampling for inference. In the first, we generated synthetic first-order rules and facts. In the second, we built a printer troubleshooting knowledge base using contributions from real users. Logistic regression was used as the evidence combination function in both experiments.

4.4.1 Synthetic Knowledge Bases

To our knowledge, there is currently no publicly-available knowledge base of the scope that would be desirable for demonstrating the advantages of our system. We thus opted to simulate the contributions of many different volunteers to a collective knowledge base, in the form of first-order rules and facts.

We based our knowledge generation process on the assumption that a contributor is an expert in a particular topic. We thus first generated a random taxonomy of topics, each of which contained some number of predicates, variable types, and ground instances. An expert is likely to know not just the concepts in a given topic, but also the general concepts of more specialized sub-topics. Each topic was thus divided into general and specific predicates. An expert could form rules for a topic[1] using as antecedents any of the topic's predicates, or any of the general predicates of the immediate sub-topics. We generated a random knowledge base of rules in this way.

We simulated an expert by choosing a random topic and sampling the knowledge base for rules with consequents from nodes in the vicinity of that topic in the hierarchy. The probability that

an expert submitted a rule in a given topic decreased exponentially with the distance (number of hops) between that topic and the expert's one in the taxonomy. We randomly added and removed antecedents from an expert's rules to simulate noisy or incomplete knowledge.

Positive training and testing examples were generated by randomly choosing a consequent and backward-chaining through rules in the knowledge base to find evidence that supported them. A positive example was turned into a negative one by removing a single evidence item, which resulted in "near-miss" examples that are easy to mistake as positive. Note that some samples thus required only knowledge contained within one topic, while others required chains of inference that spanned topics and subtopics, which we believe is often the case in the real world.

We modeled the accumulation of knowledge as proportional to the number of contributors to the system, with 25 rules and 50 feedback instances per contributor. The ontology had 25 nodes, and the "true" knowledge base had 50 rules per category. These and other parameters were constant throughout the experiments, and set before seeing any test results. We tested with 500 queries. The results are shown in Figure 4.2, where "Averaged Experts" is the performance obtained by estimating the probability of an answer as the average of the probabilities predicted by the relevant experts (i.e., those that were able to answer the question). "Trained CKB" and "Untrained CKB" refer to the performance obtained by using the architecture presented in this chapter, with or without using the feedback to train. The performance measure ("Accuracy") is the fraction of queries that were answered correctly (with unanswered queries counting as failures). The advantage of the collective knowledge base increases rapidly with the number of contributors. This is attributable to the increasing number of connections that can be made between different contributions as the knowledge base becomes increasingly densely populated. We also see that applying machine learning to estimate the quality of knowledge further improves performance.

Although not optimized for speed, our current system is fairly efficient. The time required to extract the Bayesian network for a query from the knowledge base was dominated by the time to run probabilistic inference on the network. The size of the extracted Bayesian network grew sublinearly in the size of the knowledge base, from an average of 11 nodes for a collection of 10 experts to 24 nodes for a collection of 50 experts. The average time spent on probabilistic inference for a query asked of the pool of 50 experts was 400 milliseconds[2]; the time required to run EM was approximately proportional to the product of this and the number of training examples.

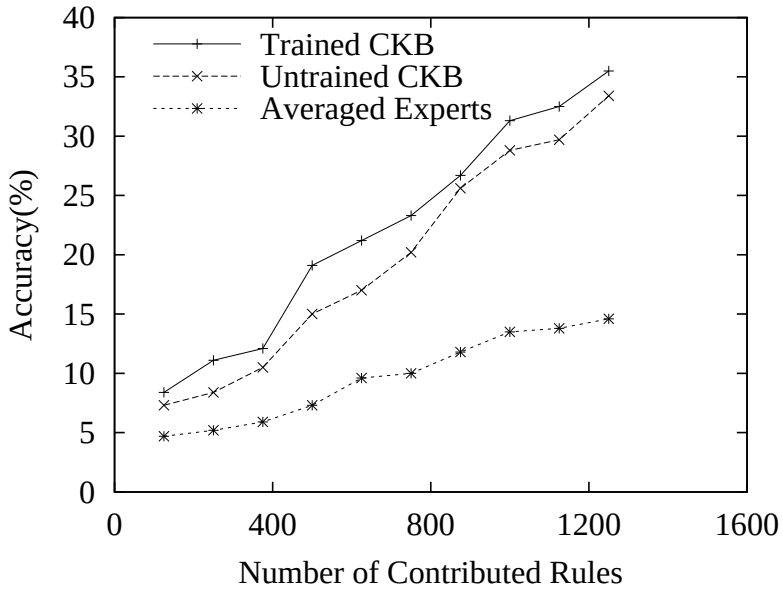


Figure 4.2: Results on synthetic knowledge bases.

4.4.2 Printer Troubleshooting

A significant portion of Usenet newsgroups, FAQs, and discussion forums is devoted to the task of helping others diagnose their computer problems. This suggests that an automated knowledge base for this domain would be in high demand. This domain is also potentially well suited to development using our collective knowledge base architecture, due to the availability of a large pool of willing experts with some degree of formal knowledge, the availability of objective outcomes for feedback purposes, the composable nature of the knowledge, the fact that evidence can potentially be captured automatically from the machines being diagnosed, etc. As a first step in this direction, we have carried out a pilot study demonstrating that knowledge obtained from real-world experts in this domain can be merged into a system that is more accurate than the experts in isolation.

We used the Microsoft printer troubleshooting Bayesian network as a model of the domain.[3] (In other words, examples generated from this network were used to simulate examples generated by the real world.) The network consists of 76 Boolean variables. Seventy of these are informational, such as “print spooling is enabled” and “fonts are installed correctly”, and six are problem-related, such as “printing takes too long”. Many of the variables are labelled as fixable and/or observable

with associated costs. We considered any variable whose cost of observation was less than one to be *evidence*, and any proposition that was fixable but not evidence to be a *cause*, which resulted in 17 evidence variables and 23 causes.

The system attempted to identify the most likely cause of a problem, given the evidence and problem nodes. To generate plausible problems a user may ask the system about, we generated random samples from the network and accepted only those where exactly one cause was at fault and at least one of the problem-related propositions was true. The system was then presented with the resulting evidence and problem nodes and asked to diagnose which proposition was the cause of the problem. As in the synthetic domain, the system may elect not to answer a question. We report two measures of success. One is the fraction of queries whose cause was properly diagnosed (with unanswered queries counting as failures). The other is the average rank of the correct diagnosis in the list of probable causes returned (with the most probable cause having rank one). This corresponds to the number of actions a user would need to perform before the printer was functioning again.

We gave the definitions of the seventy-six variables to four volunteers, who were each asked to write rules describing the printer domain to the best of their ability in a limited amount of time (see Appendix A for the directions given to the volunteers). All four were computer users who have had experience printing but did not have any particular expertise or training on the subject. Table 4.2 shows for each volunteer the time spent contributing knowledge, the number of rules contributed, and the performance before and after learning rule weights. Two hundred examples were used for training. Random guessing would have achieved an accuracy of 4.5%.

Table 4.2 also shows the results of combining the experts. The row labeled “Average” is the result of averaging predictions as described before.[4] The “CKB” row shows the added advantage of the collective knowledge base: it achieves higher accuracy than a simple combination of the individual volunteers, both when the individual volunteers’ rule coefficients have been trained and when they have not. Thus we observe once again that the collective knowledge base is able to benefit from chaining between the rules of different volunteers.

Table 4.2: Printer troubleshooting results. "Volunteer i" is the system using the ith volunteer's rules. "CKB" is the collective knowledge base. The accuracy of random guessing is 4.5%.

System	Time (mins)	Num. Rules	Accuracy (%)		Rank	
			Untrained	Trained	Untrained	Trained
Volunteer 1	120	79	11.1	15.8	11.9	6.7
Volunteer 2	30	32	2.6	4.5	9.4	8.6
Volunteer 3	120	40	2.7	10.3	13.6	10.3
Volunteer 4	60	34	3.9	6.3	13.0	12.0
Average	–	–	2.2	17.6	13.3	6.7
CKB	–	–	4.6	34.6	12.7	5.7

4.5 Limitations of KBMC

Although the architecture addresses the challenges of quality, consistency, relevance, scalability, and motivation, we found that KBMC has some undesirable limitations:

Cycles. Because KBMC uses Bayesian networks for inference, the set of Horn clauses must be acyclic. For example, the knowledge base may not have both $\texttt{A(x)} \Rightarrow \texttt{B(x)}$ and $\texttt{B(x)} \Rightarrow \texttt{A(x)}$ in it. This limitation also arises in Prolog, and can sometimes require careful knowledge engineering to surmount. Considering that our knowledge will come from multiple sources, cycles are very likely to occur. People often confuse cause and effect, writing rules "backwards" (e.g. if you have cancer, then you smoke). We found this to be a significant issue in our experiments in the printer domain: a few of the clauses entered by our experts created cycles in the knowledge base and had to be manually removed. Worse still, in many domains, there is no cause and effect, but rather implication is reasonable in either direction (e.g. both "$\texttt{Student(x)} \Rightarrow \neg\texttt{Professor(x)}$" and "$\texttt{Professor(x)} \Rightarrow \neg\texttt{Student(x)}$" are reasonable expressions). Whatever the cause, cyclic knowledge is likely to be entered to the knowledge base, and we would like to be able to handle it automatically in our reasoning framework.

Horn subset. Though much can be represented using Horn clauses, it is still only a subset of first-order logic, and thus limits what may be represented. Even many simple statements, such as "if you do not eat, you will be hungry" or "if you eat, you will either be satisfied or too full" cannot be represented (the first statement requires a negation in the antecedent and the second requires a disjunction in the consequent).

Direction of inference. Because KBMC uses Prolog and backward chaining, inference can only proceed from antecedent to consequent. That is, if we have a rule $\texttt{A}(\texttt{x}) \Rightarrow \texttt{B}(\texttt{x})$ and are able to prove $\neg\texttt{B}(\texttt{pie})$, Prolog cannot infer $\neg\texttt{A}(\texttt{pie})$. We would clearly prefer to allow inference in both directions of the implication.

Ad-hoc combination function. KBMC requires the specification of a combination function to merge the probabilities of multiple rules with the same consequent. Typically, the choice of combination function is made somewhat arbitrarily, with noisy OR seeming to be the "default" choice used by most authors. This ad-hoc selection of combination function is somewhat unsatisfying.

In the next chapter, we introduce a novel representation and inference technique, called *Markov logic networks*, which surmounts all of these limitations. The technique meets all of our representation challenges (complexity, uncertainty, modularity, comprehensibility, and inference), and allows the use of full first-order logic, making it the ideal representation for our collective knowledge base architecture.

Chapter 5

MARKOV LOGIC NETWORKS

In this chapter, we introduce *Markov logic networks* (MLNs), a novel approach that combines the full power of first-order logic and probabilistic graphical models in a single representation. From the point of view of probability, MLNs provide a compact language to specify very large Markov networks, and the ability to flexibly and modularly incorporate a wide range of domain knowledge into them. From the point of view of first-order logic, MLNs add the ability to soundly handle uncertainty, tolerate imperfect and contradictory knowledge, and reduce brittleness. MLNs fulfill all of our representational desiderata, without any of the limitations of KBMC (Section 4.5). They also provide a unifying framework for statistical relational learning (SRL).

5.1 Markov Logic

Markov logic is a simple yet powerful combination of Markov networks and first-order 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 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.

We call a set of formulas in Markov logic a *Markov logic network* or MLN. In this chapter, we make the assumption that we are in a finite domain. Extending Markov logic networks to infinite domains is a topic of future work. MLNs define probability distributions over possible worlds [57] as follows.

Definition 5.1.1 *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 $M_{L,C}$ (Equation 2.3) 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.*

Remarks:

1. The graphical structure of $M_{L,C}$ follows from the above definition: 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. Thus, the predicates in each ground formula form a (not necessarily maximal) clique in $M_{L,C}$.

2. An MLN can be viewed as a *template* for constructing Markov networks. In different worlds (different sets of constants) it will produce different networks, and these may be of widely varying size, but all will have certain regularities in structure and parameters, given by the MLN (e.g., all groundings of the same formula will have the same weight). We call each of these networks a *ground Markov network* to distinguish it from the first-order MLN.

3. When constants and variables are typed, variables are only grounded to constants of the same type. Typing can vastly reduce the size of ground Markov networks.

4. When a function appears in a formula, the value of the function will appear in groundings of the formula (and corresponding features). For example, if $\texttt{Friend}(\texttt{x}, \texttt{MotherOf}(\texttt{x}))$ is a formula in the MLN and $\texttt{MotherOf}(\texttt{Bob}) = \texttt{Anna}$, the ground network will have a feature for the formula $\texttt{Friend}(\texttt{Bob}, \texttt{Anna})$.

5. When a clause contains Skolem functions, each corresponding ground clause contains all the literals formed by replacing each Skolem function with each constant in the domain (i.e.,

the disjunction implicit in existential quantification is made explicit). For example, if $C = \{\texttt{Anna}, \texttt{Bob}\}$, the unit clause $\texttt{Friend}(\texttt{x}, \texttt{g}(\texttt{x}))$ with Skolem function $\texttt{g}(\texttt{x})$ (meaning "Everyone has a friend") yields the two ground clauses $\texttt{Friend}(\texttt{Anna}, \texttt{Anna}) \vee \texttt{Friend}(\texttt{Anna}, \texttt{Bob})$ and $\texttt{Friend}(\texttt{Bob}, \texttt{Anna}) \vee \texttt{Friend}(\texttt{Bob}, \texttt{Bob})$. If constants are typed, only constants of the function's output type are used.

6. An infinite number of constants ($|C| = \infty$) would lead to an infinite Markov network. We believe our definition and algorithms for MLNs can be generalized to this case (see Jaeger [65]), but this is an issue of chiefly theoretical interest, and we leave it for future work.

7. In logic, a KB has the same meaning (i.e., it defines the same set of possible worlds) irrespective of how it is divided into individual formulas. Similarly, in the limit of infinite weights, the distribution defined by an MLN does not depend on how it is broken up into formulas. When the weights are not infinite, the MLN provides flexibility in defining the shape of the probability distribution. The shape of the distribution depends on, among other things, how the KB is split in to formulas (recall, each formula gets one weight, so this decision amounts to selecting how many probabilistic parameters the model will have). At one extreme, we can view the entire KB as one formula, and the MLN assigns the same probability to all worlds inconsistent with it. At the other extreme, we can convert the KB into clausal form, and take each clause as a formula. This may be a good default strategy, since it allows the greatest flexibility in specifying distributions over worlds, and the most gradual decay in probability as worlds diverge from the KB. This flexibility is a feature of MLNs that is lacking in some other methods for combining logic and probability.

8. In practice, we have found it useful to add each predicate to the MLN as a unit clause. Roughly speaking, the weight of a unit clause can capture the marginal distribution of the corresponding predicate, leaving the weights of the non-unit clauses free to model only dependencies between predicates.

9. An MLN without variables (i.e., containing only ground formulas) is an ordinary Markov network. Any log-linear model over Boolean variables can be represented as an MLN, since

Table 5.1: Example of a Markov logic network. $\texttt{Fr}()$ is short for $\texttt{Friends}()$, $\texttt{Sm}()$ for $\texttt{Smokes}()$, and $\texttt{Ca}()$ for $\texttt{Cancer}()$.

English	Clausal Form	Weight
Friends of friends are friends.	$\neg\texttt{Fr}(\texttt{x},\texttt{y}) \vee \neg\texttt{Fr}(\texttt{y},\texttt{z}) \vee \texttt{Fr}(\texttt{x},\texttt{z})$	0.7
Friendless people smoke.	$\texttt{Fr}(\texttt{x},\texttt{g}(\texttt{x})) \vee \texttt{Sm}(\texttt{x})$	2.3
Smoking causes cancer.	$\neg\texttt{Sm}(\texttt{x}) \vee \texttt{Ca}(\texttt{x})$	1.5
If two people are friends, either	$\neg\texttt{Fr}(\texttt{x},\texttt{y}) \vee \texttt{Sm}(\texttt{x}) \vee \neg\texttt{Sm}(\texttt{y})$,	1.1
both smoke or neither does.	$\neg\texttt{Fr}(\texttt{x},\texttt{y}) \vee \neg\texttt{Sm}(\texttt{x}) \vee \texttt{Sm}(\texttt{y})$	1.1

each state of a Boolean clique is defined by a conjunction of literals. (This extends trivially to discrete variables, and to binary encoding of numeric variables.)

10. As weights increase, an MLN increasingly resembles a purely logical KB. In the limit of all infinite weights, the MLN represents a uniform distribution over the worlds that satisfy the KB. (A non-uniform distribution could easily be represented using additional formulas with non-zero weights.)

11. If a knowledge base *KB* is satisfiable, the satisfying assignments are the modes of the distribution represented by an MLN consisting of *KB* with all positive weights.

12. Unlike an ordinary first-order KB, an MLN can produce useful results even when it contains contradictions. An MLN can also be obtained by merging several KBs, even if they are partly incompatible. This is a crucial feature for a representation to be used in a collective knowledge base.

A first-order KB can be transformed into an MLN simply by assigning a weight to each formula.[1] For example, by adding weights to each formula of the KB from Chapter 2 (Table 2.1), we obtain Table 5.1; the last two columns of which constitute an MLN. For the set of constants $C = \{\texttt{Anna}, \texttt{Bob}\}$ ($\{\texttt{A}, \texttt{B}\}$ for short), the resulting Markov network $M_{L,C}$ is shown in Figure 5.1 (For simplicity of exposition, we omit the effect of the first two clauses).

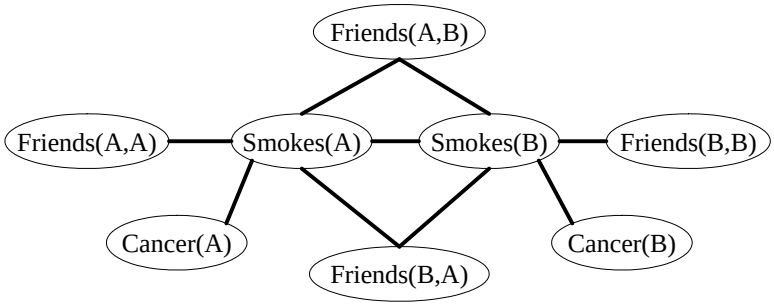


Figure 5.1: Example ground Markov network $M_{L,C}$ where L is given by the last two rows in Table 5.1, and $C = \{\texttt{Anna}, \texttt{Bob}\} = \{\texttt{A}, \texttt{B}\}$.

According to the MLN in Table 5.1, other things being equal, a world where n friendless people are non-smokers is $e^{2.3n}$ times less probable than a world where everyone has friends. Notice that all the formulas in Table 5.1 are false in the real world as universally quantified logical statements, but capture useful information on friendships and smoking habits, when viewed as features of a Markov network. For example, it is well known that teenage friends tend to have similar smoking habits [86]. In fact, an MLN like the one in Table 5.1 succinctly represents a type of model that is a staple of social network analysis [125]. This flexibility is crucial for our collective knowledge base system. If we required rules to be absolutely true, then it would be nearly impossible to model any non-trivial, real-world domain.

Markov logic networks thus satisfy our desire for a representation that handles complexity and uncertainty. They are also clearly modular (being based on individual first-order statements) and fairly comprehensible. Later in this chapter, we will demonstrate how inference in them may be carried out efficiently. Besides satisfying our desiderata for a knowledge representation for collective knowledge bases, MLNs are powerful enough to have broad appeal. In the next two sections, we show how popular SRL approaches like probabilistic relational models, knowledge-based model construction and stochastic logic programs are special cases of Markov logic. We also show how standard SRL tasks like collective classification, link prediction, link-based clustering, social network modeling, and object identification can be concisely formulated in Markov logic.

5.2 Relation to First-Order Logic

In this section, we show that for any given consistent, first-order knowledge base, it is possible to construct a Markov logic network that, for any set of constants C, has the following properties in the limit of infinite weights: (a) any formula entailed by the knowledge base has probability 1, and (b) any world that does not satisfy the knowledge base has probability 0. We begin with some simple definitions:

Definition 5.2.1 *Given a conjunction of first-order formulas $F = \{f_1 \wedge f_2 \wedge ... \wedge f_N\}$, and a set of constants C, let $G(F, C)$ be the set of all possible ground formulas obtained by grounding each F_i using the constants in C.*

Definition 5.2.2 *Let the **maximum consistent size** of a conjunction of ground formulas G be the maximum k such that there exists a $G_c \subseteq G$ with $|G_c| = k$ and at least one world satisfies the conjunction of formulas contained in G_c.*

Trivially, every conjunction of formulas has some such k (at worse, $k = 0$. At best, $k = |G|$).

Definition 5.2.3 *Let KB be a knowledge base that is a conjunction of first-order formulas. Then we define $MLN(KB, \beta)$ to be the Markov logic network defined by KB with a weight of β on each formula.*

The following lemma gives properties for Markov logic networks with infinite weights, built from possibly inconsistent knowledge bases:

Lemma 5.2.4 *Let KB be a knowledge base that is a conjunction of first-order formulas $F = \{f_1, f_2, ..., f_N\}$, and C be a set of constants, and G be the resulting set of ground formulas $G = G(F, C)$. Let k be the maximum consistent size of G. Then $M_{MLN(Kb,\beta),C}$ defines the unique probability distribution $P(X)$ that gives equal, non-zero probability to every world that satisfies k of the formulas in G, and, as $\beta \to \infty$, zero probability to every other world.*

Proof of Lemma 5.2.4: Let X_k represent a world that satisfies k formulas in G, and $X_{\neq k}$ represent a world that satisfies some number other than k formulas. Then $P(X_k) = \frac{1}{Z} \exp(k\beta)$ (which implies

that every X_k has equal, non-zero probability) and $P(X_{\neq k}) = \frac{1}{Z}\exp((k-\epsilon)\beta)$. We know that $\epsilon > 0$ because any $X_{\neg k}$ must satisfy less than k formulas (since k is the maximum consistent size). $\frac{P(X_{\neq k})}{P(X_k)} = \exp(-\epsilon\beta) \to 0$ as $\beta \to \infty$. Since $\forall_{x \in \mathcal{X}} P(x) \geq 0$ and $\sum_{x \in \mathcal{X}} P(x) = 1$, this implies that $P(X_{\neq k}) \to 0$ and $P(X_k) > 0$ as $\beta \to \infty$.

When the knowledge base is consistent, we derive the following Theorem:

Theorem 5.2.5 *Let KB be a consistent knowledge base, and $P(X)$ be the probability distribution defined by $M_{MLN(KB,\beta),C}$. Also, define $P(f)$ to be the total probability of all worlds that satisfy the first-order formula f. If KB is consistent, then:*

1. *Any world that satisfies KB has equal, non-zero probability. Any other world approaches probability zero as $\beta \to \infty$.*

2. *If $KB \models f$ then $P(f) \to 1$ as $\beta \to \infty$.*

Proof of Theorem 5.2.5:

1. Since KB is consistent, the maximum consistent size is the number of ground formulas in $G(KB, C)$. Thus, from Lemma 5.2.4, $P(X)$ gives equal, non-zero probability to any world that satisfies every formula in $G(KB, C)$, and, as $\beta \to \infty$, zero probability to every other world. This directly implies that $P(X)$ gives equal, non-zero probability to any world that satisfies KB, and, as $\beta \to \infty$, zero probability to every other world.

2. Let W_{KB} be the set of worlds that satisfy KB, W_f be the set of worlds that satisfy f, and W the set of all possible worlds. Since $KB \models f$, we know that $W_{KB} \subseteq W_f$, and therefore $\sum_{w \in W_{KB}} P(w) \leq \sum_{w \in W_f} P(w)$. Recall that $\sum_{w \in W} P(w) = 1$. From Theorem 5.2.4, Part 1, as $\beta \to \infty$ $\sum_{w \notin W_{KB}} P(w) \to 0$, which implies that $\sum_{w \in W_{KB}} P(w) \to 1$, and therefore $\sum_{w \in W_f} P(w) \to 1$

5.3 Markov Logic Subsumes SRL Approaches

Since Markov logic subsumes first-order logic and probabilistic graphical models, it subsumes all representations used in SRL that are formed from special cases of them. It is enlightening to see

how these representations map into Markov logic, and here we informally do this for a few of the most popular ones, which we previously surveyed in Chapter 3.

5.3.1 Knowledge-Based Model Construction

A KBMC model is translated into Markov logic by writing down a set of formulas for each first-order predicate $\texttt{Pk}(...)$ in the domain. Each formula is a conjunction containing $\texttt{Pk}(...)$ and one literal per parent of $\texttt{Pk}(...)$ (i.e., per first-order predicate appearing in a Horn clause having $\texttt{Pk}(...)$ as the consequent). A subset of these literals are negated; there is one formula for each possible combination of positive and negative literals. The weight of the formula is $w = \log[p/(1-p)]$, where p is the conditional probability of the child predicate when the corresponding conjunction of parent literals is true, according to the combination function used. If the combination function is logistic regression, it can be represented using only a linear number of formulas, taking advantage of the fact that it is a (conditional) Markov network with a binary clique between each predictor and the response. Noisy OR can similarly be represented with a linear number of parents.

MLNs have several advantages compared to KBMC: they allow arbitrary clauses (not just Horn ones) and inference in any direction, they sidestep the thorny problem of avoiding cycles in the Bayesian networks constructed by KBMC, and they do not require the introduction of *ad hoc* combination functions for clauses with the same consequent.

5.3.2 Stochastic Logic Programs

It has been shown that SLPs are a special case of KBMC [108]. Thus, they can be represented in Markov logic in the same way.

5.3.3 Probabilistic Relational Models

PRMs can be represented in Markov logic by defining a predicate $\texttt{S}(\texttt{x}, \texttt{v})$ for each (propositional or relational) attribute of each class, where $\texttt{S}(\texttt{x}, \texttt{v})$ means "The value of attribute $\texttt{S}$ in object $\texttt{x}$ is $\texttt{v}$." A PRM is then translated into Markov logic by writing down a formula for each line of each (class-level) conditional probability table (CPT) and value of the child attribute. The formula is a conjunction of literals stating the parent values and a literal stating the child value, and its weight is

the logarithm of $P(x|Parents(x))$, the corresponding entry in the CPT.[2] In PRMs, each attribute must take exactly one value, so the MLN also contains formulas with infinite weights stating this. Notice that this approach handles all types of uncertainty in PRMs (attribute, reference and existence uncertainty).

As Taskar et al. [122] point out, the need to avoid cycles in PRMs causes significant representational and computational difficulties. Also, PRMs require specifying a complete conditional model for each attribute of each class, which in large complex domains can be quite burdensome. In contrast, MLNs create a complete joint distribution from whatever number of first-order features the user chooses to specify.

5.3.4 Relational Markov Networks

An RMN is simply an MLN with a formula (in particular, a conjunction of literals) for each possible state of each clique template in the RMN, with the corresponding weight. RMNs use all states of a clique as features, which makes them exponential in clique size. MLNs can be viewed as improving the scalability of RMNs by also specifying which features to use (namely, clauses defined on the clique variables).

5.3.5 Structural Logistic Regression

In structural logistic regression (SLR) [106], the predictors are the output of SQL queries over the input data. Just as a logistic regression model is a discriminatively-trained Markov network, an SLR model is a discriminatively-trained MLN.[3]

5.3.6 Relational Dependency Networks

In a relational dependency network (RDN), each node's probability conditioned on its Markov blanket is given by a decision tree [96]. Every RDN has a corresponding MLN in the same way that every dependency network has a corresponding Markov network, given by the stationary distribution of a Gibbs sampler operating on it [60].

5.3.7 Plates

Large graphical models with repeated structure are often compactly represented using plates [54]. MLNs subsume plates as a representation language. In addition, they allow individuals and their relations to be explicitly represented (see Cussens [24]), and context-specific independencies to be compactly written down, instead of left implicit in the node models.

5.4 Markov Logic Handles Key SRL Tasks

In this section, we show how key SRL tasks can be concisely formulated in Markov logic, making it possible to bring the full power of logical and statistical learning and inference approaches to bear on them.

5.4.1 Collective Classification

The goal of ordinary classification is to predict the class of an object given its attributes. In collective classification, we also take into account the classes of related objects. Attributes can be represented in Markov logic as predicates of the form $\mathtt{A}(\mathtt{x},\mathtt{v})$, where $\mathtt{A}$ is an attribute, $\mathtt{x}$ is an object, and $\mathtt{v}$ is the value of $\mathtt{A}$ in $\mathtt{x}$. The class is a designated attribute $\mathtt{C}$, representable by $\mathtt{C}(\mathtt{x},\mathtt{v})$, where $\mathtt{v}$ is $\mathtt{x}$'s class. Classification is now simply the problem of inferring the truth value of $\mathtt{C}(\mathtt{x},\mathtt{v})$ for all $\mathtt{x}$ and $\mathtt{v}$ of interest given all known $\mathtt{A}(\mathtt{x},\mathtt{v})$. Ordinary classification is the special case where $\mathtt{C}(\mathtt{xi},\mathtt{v})$ and $\mathtt{C}(\mathtt{xj},\mathtt{v})$ are independent for all $\mathtt{xi}$ and $\mathtt{xj}$ given the known $\mathtt{A}(\mathtt{x},\mathtt{v})$. In collective classification, the Markov blanket of $\mathtt{C}(\mathtt{xi},\mathtt{v})$ includes other $\mathtt{C}(\mathtt{xj},\mathtt{v})$, even after conditioning on the known $\mathtt{A}(\mathtt{x},\mathtt{v})$. Relations between objects are represented by predicates of the form $\mathtt{R}(\mathtt{xi},\mathtt{xj})$. A number of interesting generalizations are readily apparent, for example $\mathtt{C}(\mathtt{xi},\mathtt{v})$ and $\mathtt{C}(\mathtt{xj},\mathtt{v})$ may be indirectly dependent via unknown predicates, possibly including the $\mathtt{R}(\mathtt{xi},\mathtt{xj})$ predicates themselves. Background knowledge can be incorporated by stating it in first-order logic, learning weights for the resulting formulas, and possibly refining them.

5.4.2 Link Prediction

The goal of link prediction is to determine whether a relation exists between two objects of interest (e.g., whether `Anna` is `Bob`'s Ph.D. advisor) from the properties of those objects and possibly other

known relations. The formulation of this problem in Markov logic is identical to that of collective classification, with the only difference that the goal is now to infer the value of $\mathtt{R}(\mathtt{xi}, \mathtt{xj})$ for all object pairs of interest, instead of $\mathtt{C}(\mathtt{x}, \mathtt{v})$.

5.4.3 Link-Based Clustering

The goal of clustering is to group together objects with similar attributes. In model-based clustering, we assume a generative model $P(X) = \sum_C P(C)P(X|C)$, where X is an object, C ranges over clusters, and $P(C|X)$ is X's degree of membership in cluster C. In link-based clustering, objects are clustered according to their links (e.g., objects that are more closely related are more likely to belong to the same cluster), and possibly according to their attributes as well. This problem can be formulated in Markov logic by postulating an unobserved predicate $\mathtt{C}(\mathtt{x}, \mathtt{v})$ with the meaning "$\mathtt{x}$ belongs to cluster $\mathtt{v}$," and having formulas in the MLN involving this predicate and the observed ones (e.g., $\mathtt{R}(\mathtt{xi}, \mathtt{xj})$ for links and $\mathtt{A}(\mathtt{x}, \mathtt{v})$ for attributes). Link-based clustering can now be performed by learning the parameters of the MLN, and cluster memberships are given by the probabilities of the $\mathtt{C}(\mathtt{x}, \mathtt{v})$ predicates conditioned on the observed ones.

5.4.4 Social Network Modeling

Social networks are graphs where nodes represent social actors (e.g., people) and arcs represent relations between them (e.g., friendship). Social network analysis [125] is concerned with building models relating actors' properties and their links. For example, the probability of two actors forming a link may depend on the similarity of their attributes, and conversely two linked actors may be more likely to have certain properties. These models are typically Markov networks, and can be concisely represented by formulas like $\forall \mathtt{x} \forall \mathtt{y} \forall \mathtt{v}\ \mathtt{R}(\mathtt{x}, \mathtt{y}) \Rightarrow (\mathtt{A}(\mathtt{x}, \mathtt{v}) \Leftrightarrow \mathtt{A}(\mathtt{y}, \mathtt{v}))$, where $\mathtt{x}$ and $\mathtt{y}$ are actors, $\mathtt{R}(\mathtt{x}, \mathtt{y})$ is a relation between them, $\mathtt{A}(\mathtt{x}, \mathtt{v})$ represents an attribute of $\mathtt{x}$, and the weight of the formula captures the strength of the correlation between the relation and the attribute similarity. For example, a model stating that friends tend to have similar smoking habits can be represented by the formula $\forall \mathtt{x} \forall \mathtt{y}\ \mathtt{Friends}(\mathtt{x}, \mathtt{y}) \Rightarrow (\mathtt{Smokes}(\mathtt{x}) \Leftrightarrow \mathtt{Smokes}(\mathtt{y}))$. Notice that this formula is false as a universally quantified statement in first-order logic, but is true in some domains as a probabilistic statement in Markov logic [86]. As well as encompassing existing social network models, Markov

logic allows richer ones to be easily stated (e.g., by writing formulas involving multiple types of relations and multiple attributes, as well as more complex dependencies between them).

5.4.5 Object Identification

Object identification (also known as record linkage, de-duplication, and others) is the problem of determining which records in a database refer to the same real-world entity (e.g., which entries in a bibliographic database represent the same publication). This problem is of crucial importance to many companies, government agencies, and large-scale scientific projects. One way to represent it in Markov logic is by defining a predicate $\texttt{Same}(\texttt{x}, \texttt{y})$ with the meaning "$\texttt{x}$ represents the same real-world entity as $\texttt{y}$." This predicate is applied both to records and their fields (e.g., $\texttt{Same}(\text{"ICML"}, \text{"Intl. Conf. on Mach. Learn."})$). The dependencies between record matches and field matches can then be represented by formulas like $\forall \texttt{x} \forall \texttt{y}\ \texttt{Same}(\texttt{x}, \texttt{y}) \Leftrightarrow \texttt{Same}(\texttt{fi}(\texttt{x}), \texttt{fi}(\texttt{y}))$, where $\texttt{x}$ and $\texttt{y}$ are records and $\texttt{fi}(\texttt{x})$ is a function returning the value of the $\texttt{i}$th field of record $\texttt{x}$. We have successfully applied this approach to de-duplicating the Cora database of computer science papers [100]. Because it allows information to propagate from one match decision (i.e., one grounding of $\texttt{Same}(\texttt{x}, \texttt{y})$) to another via fields that appear in both pairs of records, it effectively performs collective object identification, and in our experiments outperformed the traditional method of making each match decision independently of all others. For example, matching two references may allow us to determine that "ICML" and "MLC" represent the same conference, which in turn may help us to match another pair of references where one contains "ICML" and the other "MLC." Markov logic also allows additional information to be incorporated into a de-duplication system easily, modularly and uniformly. For example, transitive closure is incorporated by adding the formula $\forall \texttt{x} \forall \texttt{y} \forall \texttt{z}\ \texttt{Same}(\texttt{x}, \texttt{y}) \wedge \texttt{Same}(\texttt{y}, \texttt{z}) \Rightarrow \texttt{Same}(\texttt{x}, \texttt{z})$, with a weight that can be learned from data.

5.5 Implementation

Probabilistic inference and first-order logical inference in a finite domain are both intractable, and therefore the same is true of inference in MLNs. However, many of the large number of techniques for efficient inference in either case are applicable to MLNs. Because MLNs allow fine-grained encoding of knowledge, including context-specific independencies, inference in them may in some

cases be more efficient than inference in an ordinary graphical model for the same domain. On the logic side, the probabilistic semantics of MLNs makes approximate inference possible, with the corresponding potential gains in efficiency.

In principle, any inductive logic programming (ILP) approach can be used to learn the structure of an MLN, and any approach for learning Markov network parameters (e.g., conjugate gradient or iterative scaling) can be used to learn the weights. Likewise, any method for inference in Markov networks (e.g., Markov chain Monte Carlo, belief propagation) can be used to perform inference in grounded MLNs, and logical inference methods can be used to construct the subsets of these networks relevant to a particular query.

In this section, we describe one possible implementation of Markov logic, using MaxWalkSat [119] and Gibbs sampling for inference, the CLAUDIEN ILP system [26] for structure learning, and a pseudo-likelihood method for parameter learning [11].

5.5.1 Inference

The inference algorithm proceeds in two phases, analogous to knowledge-based model construction. Given a query Q (set of ground predicates with unknown truth values)[4] and evidence E (known ground predicates, optionally via a closed world assumption), the first phase returns the minimal subset of the ground MLN required to answer the query. The algorithm for this is shown in Table 5.2. When the KB is in clausal form, the number of nodes returned can be reduced, and the algorithm sped up, by noticing that any ground clause containing a true evidence literal is always true and can be deleted beforehand. In the worst case, the algorithm returns $O(|C|^a)$ nodes, where a is the largest predicate arity in the domain (recall C is the set of constants in the domain), but in practice it may return a much smaller set. In the second phase, we apply Gibbs sampling (see Section 2.4) to the network composed of these nodes, all arcs between them in $M_{L,C}$, and the features and weights on the corresponding cliques. Because the distribution is likely to have many modes, we run the chain multiple times. To minimize burn-in time, we start each run from a mode found using MaxWalkSat, a local search algorithm for the weighted MaxSat problem (i.e., finding a truth assignment that maximizes the sum of weights of satisfied clauses).[5]

MLNs may be used for logical inference, probabilistic (propositional) inference, and probabilis-

tic first-order inference. In all three, MLNs compare favorably to existing systems:

Efficiency of Logical Inference. A MLN may be used to do purely logical reasoning by using the MaxWalkSat initialization and taking no steps of MCMC. Stochastic search algorithms such as MaxWalkSat have been shown to perform very well, even out-performing traditional resolution-based first-order reasoning techniques in planning domains[70]. Thus, using MLNs in this way compares favorably to traditional logical inference systems.

Efficiency of Propositional Probabilistic Reasoning. A MLN may be used to do probabilistic reasoning in propositional domains by simply encoding a (propositional) Markov network. Since inference is done on the resulting Markov network, it is as efficient as if a Markov network was used for the model instead of MLNs. Most common probabilistic reasoning techniques can be efficiently represented using Bayesian networks or Markov networks, and inference in Bayesian networks is sometimes done by converting them first to Markov networks. Further, special cases in which inference is simpler (e.g., see [55]) can be explicitly detected and handled appropriately. Hence, we conclude that MLNs can be as efficient as most common probabilistic reasoning techniques.

Efficiency of Statistical Relational Reasoning. Probabilistic relational models, knowledge-based model construction, and relational Markov networks are three of the most well-known SRL methods. In all three, a graphical model (a Bayesian network for the first two, a Markov network for the last) is generated on which probabilistic inference is performed. The graphical model inference is generally the most time-consuming step of the computation. The Markov network generated by MLNs will be comparable in size to the graphical models used by these three methods, since all three aim to build the minimum model necessary to answer a given query. Thus, MLNs are approximately as efficient as these three techniques.

We now show that inference on the Markov network that results from the algorithm given in Table 5.2 gives the same result as inference on the complete network $M_{L,C}$.

Proposition 5.5.1 *Let $M_{L,C}$ be a full ground Markov network (see Definition 5.1.1) and $M'_{L,C}$ be the Markov network that would be constructed by the algorithm given in Table 5.2 for the purposes*

Table 5.2: Network construction for inference in MLNs. $MB(q)$ is the Markov blanket of q in $M_{L,C}$.

Procedure ConstructNetwork(Q, E)

Let $M = Q$.

While $Q \neq \emptyset$

  For all $q \in Q$

    If $q \notin E$

      Let $Q = Q \cup (MB(q) \setminus M)$.

      Let $M = M \cup MB(q)$.

    Let $Q = Q \setminus \{q\}$.

Return M.

---

of answering a query $P(Q|E)$ (Q and E are a set of query and evidence nodes, respectively). If $P(Q|E)$ is the conditional probability distribution defined by $M_{L,C}$, and $P'_{Q,E}(Q|E)$ is the conditional probability distribution defined by $M'_{L,C}$, then $\forall_{q,e} P(Q = q|E = e) = P'_{Q,E}(Q = q|E = e)$.

In the algorithm in Table 5.1.1, Q maintains a list of the nodes which have been added to the network, but whose Markov blanket may not have been added. At each step, a node in Q is removed from Q, and its Markov blanket added to the network. This continues until Q is empty. Thus, by examination of the algorithm, it is apparent that upon completion, for every node in the constructed network, its Markov blanket is also in the constructed network (note that, the Markov blanket of an observed node is the empty set, because it is already rendered independent of the rest of the network). By the definition of Markov networks, a node is independent of the rest of the network given its Markov blanket. Since, for all nodes in M' (for exposition clarity, we drop the subscript of $M_{L,C}$), their Markov blanket is in M', we know that all nodes in M' are independent of those nodes in M that do not appear in M', and hence their probability distribution is unaffected by them.

Therefore, the probability distribution P'(Q—E)

Note that this theorem is actually stronger than we need to answer a particular query. Indeed, when the actual values of the query and evidence nodes are known, a more efficient algorithm that only adds nodes which are involved in features with unknown value may be used. This algorithm will sometimes generate a smaller network than that generated by the one given in Table 5.1.1.

5.5.2 Learning

We learn MLN weights from one or more relational databases. (For brevity, the treatment below is for one database, but the generalization to many is trivial.) If there are n possible ground predicates, a database is effectively a vector $x = (x_1, \ldots, x_l, \ldots, x_n)$ where x_l is the truth value of the lth ground predicate. By the closed world assumption, $x_l = 1$ if the lth ground predicate appears in the database, and $x_l = 0$ otherwise. Given a database, MLN weights can in principle be learned using standard methods. If the ith formula has $n_i(x)$ true groundings in the data x, then its weight w_i appears $n_i(x)$ times in Equation 2.3, and the derivative of the log-likelihood with respect to it is

$$\frac{\partial}{\partial w_i} \log P_w(X = x) = n_i(x) - \sum_{x'} P_w(X = x')\, n_i(x') \tag{5.1}$$

where the sum is over all possible databases x', and $P_w(X = x')$ is $P(X = x')$ computed using the current 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, counting the number of true groundings of a formula in a database is a #P-complete problem, even when the formula is a single clause. The proof of this (Due to Dan Suciu) is by reduction from counting satisfying assignments of monotone 2-CNF, which is #P-complete [116]. This problem can be reduced to counting the number of true groundings of a first-order formula in a database as follows. Consider a database composed of the ground predicates $R(0, 1)$, $R(1, 0)$ and $R(1, 1)$. Given a monotone 2-DNF formula, construct a formula Φ that is a conjunction of $R(x_i, x_j)$ terms, one for each disjunct in the DNF formula. (For example, $(x_1 \vee x_2) \wedge (x_3 \vee x_4)$ would yield $R(x_1, x_2) \wedge R(x_3, x_4)$. There is a one-to-one correspondence between the satisfying assignments of the 2-CNF and Φ.

Thus, in large domains, we approximate these counts by uniformly sampling groundings of a

formula, and checking whether they are true in the data. In smaller domains, we use an efficient recursive algorithm to perform the exact count.

A second problem with the equation above is that computing the expected number of true groundings is also intractable, requiring inference over the model. To surmount this difficulty, we use Monte-Carlo (MC) methods.

Monte-Carlo Maximum Likelihood

Following [51], we formulate the learning problem as one of finding the weights w which give the maximum likelihood *relative to the likelihood at a fixed set of weights* w^0 (we make $P(X)$ and Z's dependence on w explicit by notating them as $P_w(X)$ and $Z(w)$) :

$$w_{MC}(x, w^0) = \text{argmax}_w L_{MC}(x, w; w^0) \text{ , where} \tag{5.2}$$

$$L_{MC}(x, w; w^0) = \frac{P_w(x)}{P_{w^0}(x)} = \frac{Z(w^0)}{Z(w)} \exp\left(\sum_i (w_i - w_i^0) n_i(x)\right) \tag{5.3}$$

This simplifies the problem from one of estimating $Z(w)$ to one of estimating the ratio $Z(w)/Z(w^0)$. It is clear that, since $P_{w^0}(x)$ is constant, $w_{MC}(x, w^0)$ is the maximum likelihood estimate. To evaluate Equation 5.3,we need to compute $\frac{Z(w)}{Z(w_0)}$, which can be done using Monte-Carlo sampling:

$$\frac{Z(w)}{Z(w^0)} \approx \frac{1}{m} \sum_{t=1}^{m} \exp\left(\sum_i (w_i - w_i^0) n_i(x^t)\right) \tag{5.4}$$

where x^t is the tth sample taken from the distribution $P_{w^0}(X)$. The approximation error goes to zero as m goes to infinity. Combining Equations 5.3 and 5.4 and taking the logarithm, we get

$$\log L_{MC}(x, w; w^0) = - \log\left[\frac{1}{m} \sum_{t=1}^{m} \exp\left(\sum_i (w_i - w_i^0)(n_i(x^t) - n_i(x))\right)\right] \tag{5.5}$$

The gradient is

$$\frac{\partial}{\partial w_i} \log L_{MC}(x, w; w^0) = \tag{5.6}$$

$$= -\frac{\sum_{t=1}^{m} \exp\left(\sum_i (w_i - w^0)(n_i(x^t) - n_i(x))\right)(n_i(x^t) - n_i(x))}{\sum_{t=1}^{m} \exp\left(\sum_i (w_i - w^0)(n_i(x^t) - n_i(x))\right)} \tag{5.7}$$

Standard gradient ascent methods may be used to find the optimal $w_{MC}(x, w^0)$. However, note that many samples may be required when w^0 is not near the true MLE. Hence, an iterative procedure is often employed, beginning with some w^0, and incrementally finding $w^r = w_{MC}(x, w^{r-1})$ until convergence. One problem with this is that the absolute magnitude of $L_{MC}(w, x; w^r)$ keeps changing as the w^r changes (the 0-point is at $w = w^r$). This may be corrected by subtracting $\log L_{MC}(x, w^0; w^r)$, since $\log L_{MC}(x, w; w^0) = \log L_{MC}(x, w; w^r) - \log L_{MC}(x, w^0; w^r)$. However, as w^r moves away from w^0, the approximation of $\log L_{MC}(x, w^0; w^r)$ becomes worse. Instead, we use *bridge sampling* [44]:

$$\log L_{MC}(x, w; w^0) = \log L_{MC}(x, w; w^r) - \sum_{j=0}^{r} \log L_{MC}(x, w^{j-1}; w^j) \tag{5.8}$$

In this way, samples taken using w^r are used only to estimate $\log L_{MC}(x, w^{r-1}; w^r)$ and $\log L_{MC}(x, w^{r+1}; w^r)$. It can also sometimes be useful to limit the distance between w^r and w^{r+1} to reduce the error in the approximation of L_{MC}.

The summation in Equation 5.8 can be updated incrementally, which leads to the following algorithm:

Select an initial set of weights, w^0

$r = 0$, $c = 0$

While not converged

  Generate samples $x^t \sim P_{w^r}(x)$

  Find $w^{r+1} = \text{argmax}_w \log L_{MC}(x, w; w^r) - c$

  $c = c + \log L_{MC}(x, w^r, w^{r+1})$

  $r = r + 1$

The computational complexity is the same as it would be without bridge sampling, but in our experience provides significant improvements in accuracy. Also note, some minimization methods, such as conjugate gradient with line minimization, are unaffected by a change in the 0-point of the function being minimized, so may be used without computing c. In our implementation, we use limited-memory Broyden-Fletcher-Goldfarb-Shanno [84](L-BFGS) to find the MLE. The samples

are generated using MCMC (specifically, Gibbs sampling). To combat overfitting, we optionally penalize the pseudo-likelihood with a Gaussian prior on each weight.

Finding the MLE in this way can be computationally expensive. A more efficient alternative, widely used in areas like spatial statistics, social network modeling and language processing, is to optimize instead the pseudo-likelihood.

Pseudo-likelihood

The pseudo-likelihood is defined as [11]:

$$P_w^*(X=x) = \prod_{l=1}^{n} P_w(X_l=x_l|MB_x(X_l)) \tag{5.9}$$

where $MB_x(X_l)$ is the state of the Markov blanket of X_l in the data. The gradient of the pseudo-log-likelihood is

$$\frac{\partial}{\partial w_i} \log P_w^*(X=x) = \sum_{l=1}^{n} [n_i(x) - P_w(X_l=0|MB_x(X_l))\, n_i(x_{[X_l=0]}) - P_w(X_l=1|MB_x(X_l))\, n_i(x_{[X_l=1]})] \tag{5.10}$$

where $n_i(x_{[X_l=0]})$ is the number of true groundings of the ith formula when we force $X_l = 0$ and leave the remaining data unchanged, and similarly for $n_i(x_{[X_l=1]})$. Computing this expression does not require inference over the model. The sum can be computed efficiently by ignoring predicates that do not appear in the ith formula. Counts need only be computed once, and clause groundings that contain at least two true literals in the data (typically the great majority) can be ignored (since then $n_i(x_{[X_l=0]}) = n_i(x_{[X_l=1]}) = n_i(x)$ for all l). Because weights are shared among all groundings of a clause, each iteration of the search (after the initial one) requires only $O(n * (\#\text{ of first order clauses}))$ computation, a significant improvement over $O(\#\text{ of ground clauses})$, the number of features in the Markov network. As with MCMLE, we optimize the pseudo-log-likelihood using L-BFGS, optionally penalizing the pseudo-likelihood with a Gaussian prior on each weight.

Although finding the maximum pseudo-likelihood is is less computationally expensive, it may give bad estimates in situations where there are many strong dependencies in the data. In some situations, the pseudo-likelihood is degenerate, and will lead to unpredictable results [58]. Experimentally, we found this to be a problem primarily when the model is trained to a very low

convergence threshold. To combat this overtraining, we do "early-stopping" by selecting, out of the models generated by the training process, the one that has approximately the best likelihood on the training data.

More specifically, let w^r, $r \in \{1..k\}$ be the model parameters found in the rth iteration of optimizing the pseudo-likelihood. We would like to select the w^r that maximizes the likelihood of the training data: $P_{w^r}(x)$. Unfortunately, as discussed in Section 5.5.2, this is difficult. Instead, we compute all the pairwise relative log-likelihoods: $\mathbf{S}_{ij} = L_{MC}(x, w^j; w^i)$ (see Equation 5.5). We cannot directly compare one row of $\mathbf{S}$ to another, since they are taken from different sets of samples. However, the elements in a row are all taken from the same set of samples, so we convert each row into a rank-vector, giving the ranking of each w^j in the row. The w^r with best average rank across all rows is chosen as the final model. Note that computing $\mathbf{S}_{ij}$ requires sampling k times, once per row of the matrix. This can be reduced by considering only a subset of w^r (in our experiments, we consider only every 10^{th} one). Also, one needs only enough samples to distinguish between a good model and a poor one, which should be (and was confirmed empirically to be true in our experiments) much less than the number of samples needed for methods such as MCMLE.

Structure learning

ILP techniques can be used to learn additional clauses, refine the ones already in the MLN, or learn an MLN from scratch. Currently we use the CLAUDIEN system for this purpose [26]. Unlike most other ILP systems, CLAUDIEN is able to learn non-Horn clauses, making it well suited to MLNs. Also, by constructing a particular language bias, we are able to direct CLAUDIEN to search for refinements of the MLN structure. In the future we plan to more fully integrate structure learning into MLNs, by generalizing techniques like Della Pietra et al.'s (1997) to the first-order realm.

5.6 Experiments

We tested MLNs using a database describing the Department of Computer Science and Engineering at the University of Washington (UW-CSE). The domain consisted of 24 predicates and 2707 constants divided into 11 types. Types included: publication (342 constants), person (442), course (176), project (153), academic quarter (20), etc. Predicates included: `Professor(person)`,

Student(person), Area(x, area) (with x ranging over publications, persons, courses and projects), AuthorOf(publication, person), AdvisedBy(person, person), YearsInProgram(person, years) CourseLevel(course, level) TaughtBy(course, person, quarter), TeachingAssistant(course, person, quarter), etc.

Using typed variables, the total number of possible ground predicates (n in Subsection 5.5.2) was 4,106,841. The database contained a total of 3380 tuples (i.e., there were 3380 true ground predicates). We obtained this database by scraping pages in the department's Web site (www.cs.washington.edu). Publications and AuthorOf relations were obtained by extracting from the BibServ database (www.bibserv.org) all records with author fields containing the names of at least two department members (in the form "last name, first name" or "last name, first initial").

We obtained a collective knowledge base for this domain by asking four volunteers to write down formulas in first-order logic relating the predicates above. This yielded 95 formulas. The complete KB, volunteer instructions, database, and algorithm parameter settings can be found in Appendix B. Formulas in the KB include statements like: students are not professors; if a student is an author of a paper, so is her advisor; advanced students only TA courses taught by their advisors; each student has at most one advisor; at most one author of a given publication is a professor; etc. Notice that these statements are not always true, but are typically true.

For training and testing purposes, we divided the database into five sub-databases, one for each area: AI, graphics, programming languages, systems, and theory. Professors and courses were manually assigned to areas, and other constants were iteratively assigned to the most frequent area among other constants they appeared in some tuple with. Each tuple was assigned to the area of the constants in it. Tuples involving constants of more than one area were discarded, to avoid train-test contamination. The sub-databases contained, on average, 521 true ground predicates out of a possible 58457.

We performed leave-one-out testing by area, testing on each area in turn while training on the remaining four. The test task was to predict the AdvisedBy(x, y) predicate given (a) all others (All Info) and (b) all others except Student(x) and Professor(x) (Partial Info). In both cases, we measured the average conditional log-likelihood of all possible groundings of AdvisedBy(x, y) over all runs, drew precision/recall curves, and computed the area under the curve. Note that this task is an instance of link prediction, a problem that has been the object of much interest in the area

of statistical relational learning (e.g., Getoor & Jensen [48], etc.).

All KBs were converted to clausal form, and exact counting of clause groundings was used throughout. We used a weight prior with mean of zero and variance of 1.0. For each inference we carried out 10,000 cycles of Gibbs sampling, split into 10 restarts with a burn-in of 100 samples each. (50,000 cycles and 50 restarts with a burn-in of 500 gave essentially identical results.)

5.6.1 Comparison of Learning Methods

We first compared pseudo-likelihood (PL), pseudo-likelihood with early stopping (PL-E), and MCMLE (ML). For MCMLE, we used Gibbs sampling with 10 simultaneous chains, each initialized by letting each ground predicate be true with probability equal to the probability that the corresponding first-order predicate is true in the data. We used the maximum of the Gelman criteria [53] across formulas to determine when the chains had "forgotten" their initial state. In our case, the statistics being gathered were the number of times each first-order formula was satisfied by the sample.

In a step of Gibbs sampling, a ground predicate ρ is picked, and potentially flipped (from true to false, or vice versa). From one step to the next only formulas that unify with ρ can be affected. By ignoring the formulas that are irrelevant, this computation can be quite fast (on the UW-CSE domain, our implementation takes 15ms per step of Gibbs sampling).

Rather than selecting ρ uniformly at random, we employed the following strategy: Before sampling, mark the ground predicates that are true either in the training example or the initial state of the chain. At each step of Gibbs, with probability q uniformly select a marked predicate, and with probability $1 - q$ select an unmarked one. Preferentially selecting predicates that were true in the example or the data helps ensure that we are picking predicates that are likely to change state, since we expect most of the predicates to be false.

Despite these optimizations, the Gibbs sampler takes a prohibitively long time to reach a reasonable convergence threshold. For large thresholds (e.g., R=2), the resulting models were poor. One major difficulty is the initialization of the sample chains. In order to converge quickly, the chains should be initialized at modes of the underlying probability distribution. If the models are reasonably modelling the domain, then a good initial state may be somewhere near the training example

itself.

Our best results were achieved by initializing the chains at the training example, and sampling with no burnin period. This was motivated by the following intuition: Ideally, our model will assign a high probability to the training example. So if a Gibbs sampler starts at the training example and moves away in some direction, then such movement should be discouraged. By starting at the training example and taking a few samples, the algorithm gets an idea of the primary direction in which the chain is headed and corrects it by pushing the weights in the opposite direction.

Because pseudo-likelihood training is not based on sampling, it does not run in to the same convergence issues. It is also significantly faster, taking 20 minutes, compared to 3.8 hours for MCMLE. However, as mentioned, it can suffer from problems due to the fact that it only estimates conditional probabilities.

The results for the AllInfo case are shown in Figure 5.2. The PartialInfo case was qualitatively the same. Notice that the graph for the AI area (upper-left) shows how early stopping improved the pseudo-likelihood-trained model. The fully-trained pseudo-likelihood model is not even apparent on the graph, while the early-stopping model out-performs MCMLE.

5.6.2 Comparison to Other Systems

We have also compared MLNs to systems based on logic or probability alone. Logical inference is problematic because the KB we obtained from volunteers is inconsistent. So we do the following: Suppose we define the probability of a ground predicate in a KB as the fraction of W_k in which the ground predicate is true, where W_k is the set of worlds which satisfy k clauses, and k is the maximum consistent size of KB. In this case, in theory, we can do inference by finding the worlds which maximally satisfy KB, and count the fraction of them in which the query predicate is true. In order to approximate this, we use Gibbs sampling, initialized using walksat. The Gibbs update step is: if the new sample satisfies more clauses than the current one, then take the step (since that means the current sample should have 0 probability), if the new sample satisfies the same number of clauses then take the step with 50% probability, and if the new sample satisfies less clauses, then don't take the step. Notice that this is equivalent to a MLN built from the KB and with infinite weights.

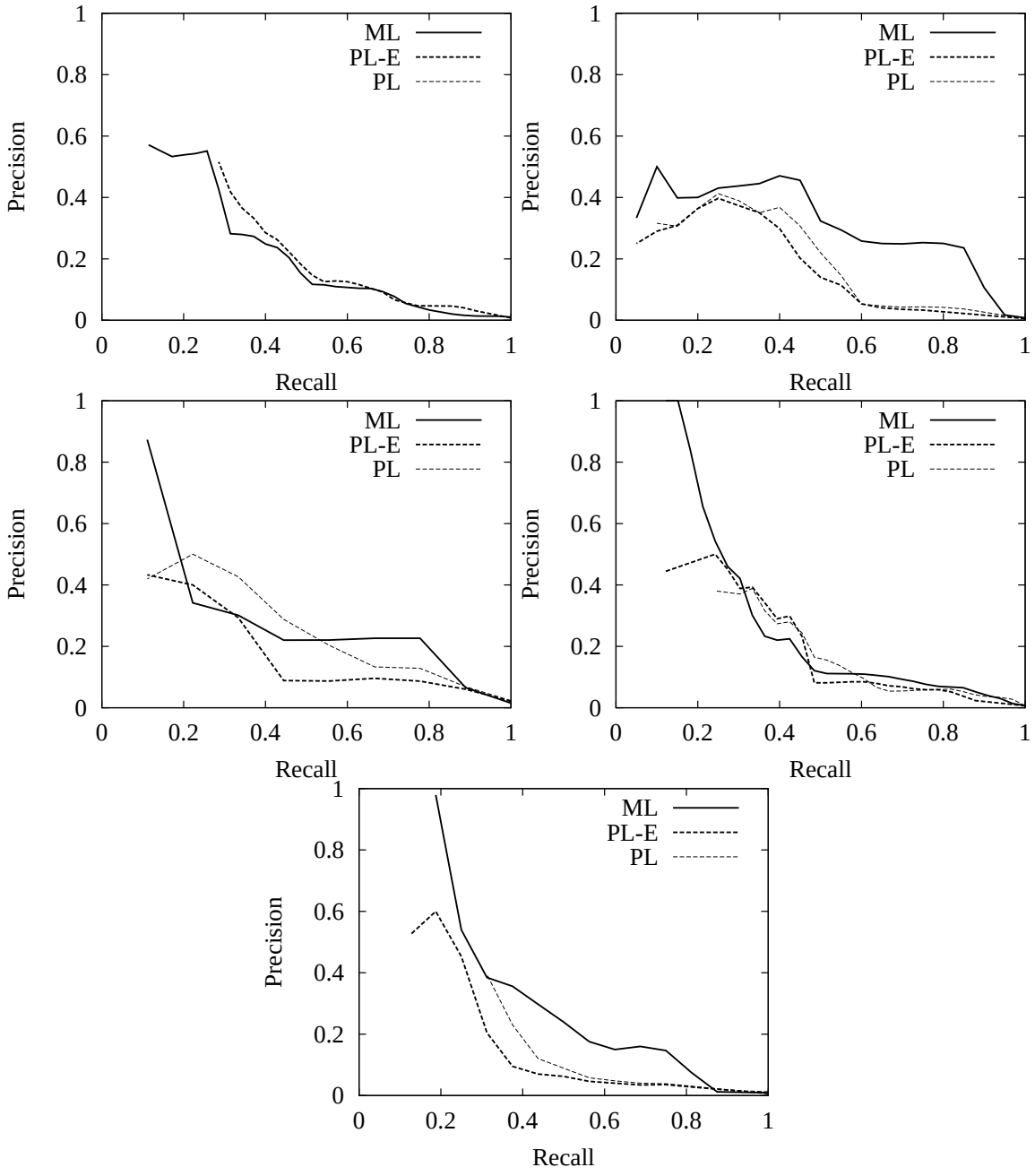


Figure 5.2: Precision and recall for `AdvisedBy`$(\mathtt{x}, \mathtt{y})$, with all other predicates known (AllInfo). Each line represents a model that was trained using a different technique: Monte-Carlo maximum likelihood (ML), pseudo-likelihood (PL), or pseudo-likelihood with early stopping (PL-E). The five graphs show the results on the five different test areas . From left to right, top to bottom, they are: AI, graphics, languages, systems, and theory.

Table 5.3: Algorithm used to generate flat attribute vectors for the propositional (Bayesian network and Naïve Bayes) learners.

To generate the attribute vector for a ground target predicate $T(t_1, t_2, \ldots, t_N)$:
For each predicate $R(r_1, r_2, \ldots, r_M)$ where $R \neq T$:
  For $i \in \{1..M\}$:
    For $j \in \{1..N\}$:
      Let **S** be the set of ground predicates where r_i=t_j.
      Add an attribute which is the fraction of **S** that are true in the data

We also compared MLNs to propositional probabilistic learners. Creating good examples for propositional learners like naive Bayes and Bayesian networks in this highly relational domain is a difficult problem. We used the following approach as a trade-off between incorporating as much potentially relevant information as possible and avoiding extremely long vectors. We defined one variable for each (a, b) pair, where a is an argument of `AdvisedBy` and b is an argument of some predicate with the same value as a. The variable is the fraction of true groundings of this predicate in the data, discretized into five equal-frequency bins. See Table 5.3 for the pseudocode. We used two propositional learners: Naive Bayes[32] and Bayesian networks. This did not work very well, so we added additional attributes: For the test predicate `AdvisedBy`(`A`, `B`) with constants `A` and `B`, we considered all pairs of predicates `Pred1`(..., `A`, ...) and `Pred2`(..., `B`, ...), and looked to see, in the cases where the non-set slots of the predicate match (i.e., the ... have the same constants in both `Pred1` and `Pred2`), how often they were both true, both false, A true while B was false, and A false while B was true. This allows the propositional learner to use attributes such as two people publishing the same paper, or one person TAing the same course that another Teaches. This added approximately 100 attributes. The extra attributes helped naïve Bayes, but hurt the performance of the Bayesian network. Below we report whichever they performed better on.

We also wanted to compare to an ILP system to see if the experts were able to contribute knowl-

edge beyond what could be found through standard machine learning techniques. We chose to use the CLAUDIEN ILP system, which induces first-order clauses rather than being restricted to first-order Horn clauses. For detailed CLAUDIEN settings, see Appendix B.

Besides inducing clauses from the training data, we also investigated using CLAUDIEN's language bias to narrow the search space to the area "near" the KB. The language bias was constructed such that, for each clause in the KB, CLAUDIEN could (1) remove any number of the literals, (2) add up to n new literals, and (3) add up to v new variables. We ran CLAUDIEN for 24 hours on a Sun-Blade 1000 with (v, n) being $(1, 2)$, $(2, 3)$, and $(3, 4)$. All three gave nearly identical results, so we used the $(3, 4)$ case.

We compared twelve systems: the original KB with logical inference (KB), CLAUDIEN induction from data with logical inference (CL), CLAUDIEN using the language bias and logical inference (CLB), logical inference on the various unions of the above systems (KB+CL and KB+CLB), and an MLN with each of these KBs (MLN(KB), MLN(CL), MLN(KB+CL), MLN(KB+CLB)), naïve Bayes (NB), and a Bayesian network learner (BN)[61],

The results obtained are summarized in Table 5.4, and precision/recall curves are shown in Figures 5.3 and 5.4. MLNs are consistently more accurate than the alternatives, showing the promise of this approach. CLAUDIEN performs poorly on its own, but can be helpful when added to the KB in the MLN. The general drop-off in precision around 30% recall is attributable to the fact that the database is very incomplete, and only allows identifying a minority of the `AdvisedBy` relations.[6] MLNs are reasonably efficient, taking on average 4 hours to learn, and 24 minutes to infer all 4900 `AdvisedBy` predicates in the Partial Info case (23 for All Info).

5.7 Summary

Markov logic networks are a representation with the characteristics required by our collective knowledge base architecture. They provide a simple way to combine probability and first-order logic without sacrificing any of the power of either. MLNs allow a wide variety of SRL tasks and approaches to be formulated in a common language, proving their general use. Experiments with knowledge contributed by volunteers demonstrated that MLNs work well as a CKB representation language.

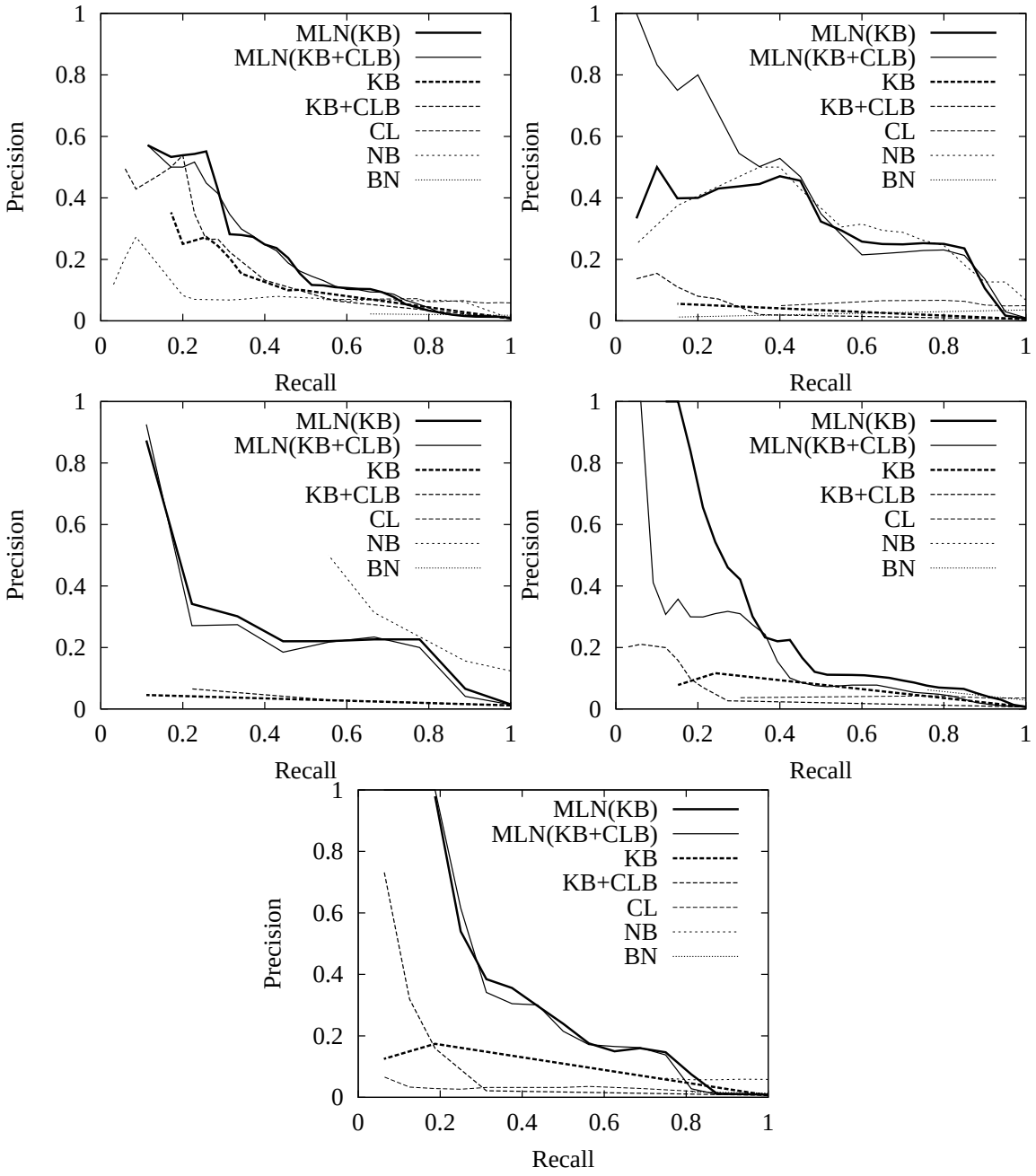


Figure 5.3: Precision and recall for AdvisedBy(x, y), with all other predicates known (All Info case). The five graphs show the results on the five different different test areas. From left to right, top to bottom, they are: AI, graphics, languages, systems, and theory.

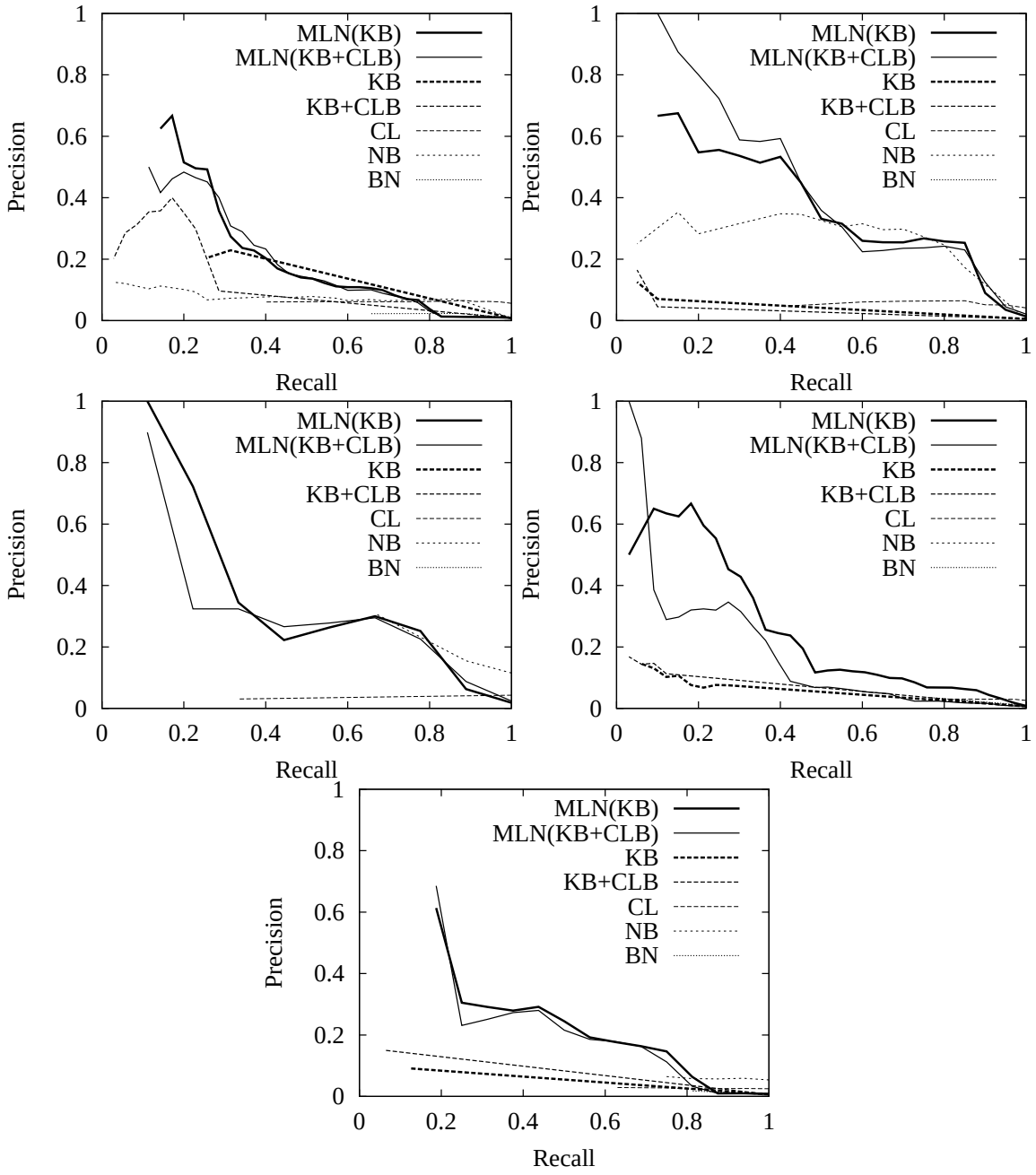


Figure 5.4: Precision and recall for AdvisedBy(x, y), with Professor(x) and Student(x) unknown (Partial Info case). The five graphs show the results on the five different different test areas. From left to right, top to bottom, they are: AI, graphics, languages, systems, and theory.

Table 5.4: Experimental results. CLL is the average conditional log-likelihood, and AUC is the area under the precision-recall curve. The results are an average over the five test sets.

System	All Info		Partial Info	
	AUC	CLL	AUC	CLL
MLN(KB)	0.310	−0.058	0.292	−0.059
MLN(KB+CLB)	0.296	−0.074	0.278	−0.075
MLN(KB+CL)	0.111	−0.768	0.086	−0.733
MLN(CLB)	0.007	−0.353	0.008	−0.691
MLN(CL)	0.034	−0.380	0.036	−0.723
KB+CLB	0.093	−0.056	0.076	−0.051
KB+CL	0.057	−0.205	0.019	−0.124
KB	0.104	−0.073	0.075	−0.290
CLB	0.007	−0.052	0.030	−0.598
CL	0.047	−0.431	0.043	−0.849
NB	0.126	−1.246	0.101	−1.170
BN	0.006	−0.177	0.010	−0.043
BN	0.032	−0.073	0.025	−0.216

Chapter 6

COLLECTIVE DETERMINATION OF DEPENDENCY STRUCTURE

Up to this point, we have looked only at knowledge in the form of logical statements. As mentioned in the introduction, we actually wish to allow contributors to provide information at all levels of detail. For instance, an expert may not know exactly how some predicates interrelate, and hence would be unable to write specific clauses in Markov logic. But the expert may be able to at least provide some information as to which predicates affect which others, without claim as to exactly how they do so (e.g. saying "smoking is somehow related to friends" instead of "if x and y are friends, then either both smoke or both don't smoke"). Such general structural information can still be very useful, as this chapter will show.

In this chapter, we present a method for learning the structure of a Bayesian network from multiple experts. Specifically, we compute the prior distribution $P(S)$ over possible structures, from expert statements. Data is then used to refine the structure and estimate parameters. A simple analysis shows that even relatively few noisy experts can produce high-quality knowledge when combined.

With both simulated and real experts, our method produces networks that are more accurate than the main alternatives: purely empirical learning, learning with knowledge from a single expert, and learning a separate model from each expert plus data and combining these. Note that this structure task is based on (propositional) Bayesian networks. Having demonstrated success in this, applying the same techniques to MLNs is the subject of future work.

6.1 Approach

Our approach is summarized in Figure 6.1. The world, assumed modelable by some Bayesian network, generates both data and the imperfect knowledge of m experts about the structure of the network. The expert knowledge (a directed graph from each expert) is used to compute a probability

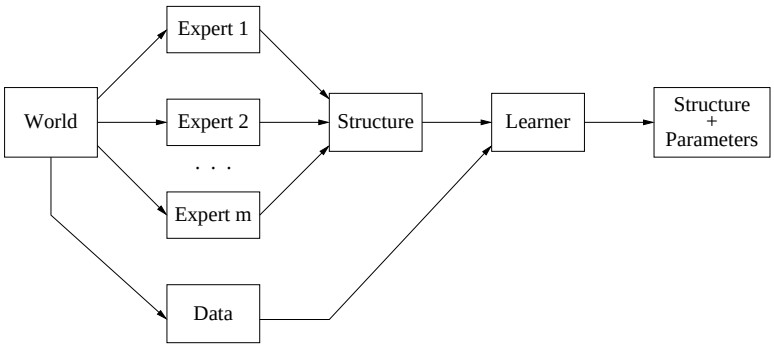


Figure 6.1: Learning with knowledge from multiple experts.

distribution over structures. Given the most probable structure and the data, we learn the parameters of the network. Optionally, given the data and the expert-induced distribution over structures, the learner finds the *a posteriori* most probable structure and the posterior distribution over parameters. (Ideally, the learner would average over all structures, but this is computationally infeasible.)

6.1.1 Basic Framework

We represent the structure of a Bayesian network with d nodes as a vector $S = (s_1, \ldots, s_j, \ldots, s_{d'})$ of $d' = d(d-1)/2$ *structure variables*, where s_j corresponds to the jth node pair in some arbitrary ordering of all the possible node pairs. For any two nodes X and Y, only one of the two pairs (X, Y) and (Y, X) is considered, the choice of which being arbitrary. Pairs of the form (X, X) are not considered. If s_j corresponds to the pair (X, Y), then $s_j = \emptyset$ if there is no arc between X and Y in the network, $s_j = \rightarrow$ if there is an arc from X to Y, $s_j = \leftarrow$ if there is an arc from Y to X, and $s_j = \leftrightarrow$ if there is an arc from to X to Y and an arc from Y to X. (Although the latter cannot happen in a real network, it may be stated by a noisy expert, and therefore we need to allow for it.)

We assume we have a pool of m experts, and the ith expert provides a vector $E_i = (e_{i,1}, \ldots, e_{ij}, \ldots, e_{id'})$, where $e_{ij} \in \{\emptyset, \rightarrow, \leftarrow, \leftrightarrow\}$ states the expert's belief about the existence of an arc between the nodes in the jth pair. Thus all the expert knowledge available is contained in the matrix $E = (E_1, \ldots, E_i, \ldots, E_m)$. In other words, each expert tells us what s/he believes are the dependencies in the domain. Because an expert's knowledge of the world is imperfect, E_i is a noisy version of S. We will often slightly abuse notation and consider E_i to "be" the ith expert.

A Bayesian network is composed of a structure S and a parameter vector Θ. Our goal is to induce a posterior distribution over (S, Θ) given expert knowledge E and a training set D. This distribution can then be used to find the most probable network (S^*, Θ^*), or to compute probabilities of interest by averaging over the possible networks, weighted by their posteriors. To make the problem tractable, we introduce a number of simplifying assumptions. By Bayes' theorem,

$$\begin{aligned} P(S, \Theta|E, D) &= \alpha P(S, \Theta) P(E, D|S, \Theta) \\ &= \alpha P(S) P(\Theta|S) P(E|S, \Theta) P(D|E, S, \Theta) \end{aligned} \tag{6.1}$$

We use α throughout to represent a normalizing constant (not necessarily always the same one). We assume that $P(E|S, \Theta) = P(E|S)$ (i.e., expert statements about structure depend only on the structure, not the parameters) and that $P(D|E, S, \Theta) = P(D|S, \Theta)$ (i.e., the data is independent of the experts given the actual structure and parameters). Substituting these equalities into Equation 6.1 and integrating both sides over Θ yields the posterior over structures

$$\begin{aligned} P(S|E, D) &= \alpha P(E|S) P(S, D) \\ &= \alpha P(S) P(E|S) P(D|S) \end{aligned} \tag{6.2}$$

where $P(S, D)$ is the standard Bayesian Dirichlet (BD) score (see Equation 2.1). The quantity $P(E|S)P(S, D)$ is the new BD score, extended to take expert statements into account by replacing the prior $P(S)$ with the "post-expert" prior $P(S|E) = \alpha P(S)P(E|S)$ (i.e., the distribution over structures after consulting the experts but before seeing any data). It can be used as the scoring function in any algorithm that learns a Bayesian network by searching over structures.

6.1.2 Expert Model

$P(E|S)$ is the generative model for expert statements, and is the key new quantity we introduce. It can be represented by a "meta-level" Bayesian network with nodes $\{s_1, \ldots, s_{d'}, e_{1,1}, \ldots, e_{1,d'}, \ldots, e_{m,1}, \ldots, e_{md'}\}$, where the s_i nodes have known values and no parents. Thus, if $par(e_{ij})$ denotes the parents of node e_{ij},

Table 6.1: Parameters of the expert model, $P(e_{ij}|s_j)$.

s_j	e_{ij}			
	$\emptyset$	$\rightarrow$	$\leftarrow$	$\leftrightarrow$
$\emptyset$	$1-2p_a-p_b$	p_a	p_a	p_b
$\rightarrow$	p_d	$1-p_d-p_r-p_c$	p_r	p_c
$\leftarrow$	p_d	p_r	$1-p_d-p_r-p_c$	p_c

$$P(E|S) = \prod_{i=1}^{m} \prod_{j=1}^{d'} P(e_{ij}|par(e_{ij})) \tag{6.3}$$

We assume the simplest useful case, which is that $par(e_{ij}) = \{s_j\}$. In other words, an expert's probability of stating that an arc exists depends only on whether the actual arc or its reverse exist, and not on what other experts state, or what the expert states about other arcs. In particular, this implies that the experts are independent given the actual structure (i.e., that they "distort reality" in independent ways; note that this is is quite different from the experts being unconditionally independent). These assumptions obviously oversimplify the behavior of real experts, but may lead to better performance than more realistic ones whose parameters would be hard to estimate.

The expert model is thus fully specified by specifying $P(e_{ij}|s_i)$ for all (i, j). We assign the same *a priori* values to these parameters for all (i, j).[1] The natural parameters to state $P(e_{ij}|s_j)$ in terms of are: p_a, the probability that the expert adds an arc in a particular direction where there was none; p_d, the probability that the expert deletes an arc; p_r, the probability that the expert reverses an arc; p_b, the probability that the expert creates a cycle (arcs in both directions) where there was no arc; and p_c, the probability that the expert creates a cycle where there was an arc. The resulting values of $P(e_{ij}|s_j)$ for all (i, j) are shown in Table 6.1. (Since $s_i = \leftrightarrow$ cannot occur, it is not necessary to specify $P(e_{ij}|s_j)$ for this case.) A simple and possibly very useful extension is to have different values of the parameters for each expert (i.e., $P(e_{ij}|s_j) = p_{ai}$, etc.).

Notice that there is nothing to preclude an expert from specifying a structure with cycles. Even if incorrect, such a structure will in general still contain useful information, and our method allows

us to extract it.

We also need to specify a prior $P(S)$ over structures. In this thesis we assume that each pair of nodes independently has some probability p_0 of being connected by an arc in a given direction. We also know that Bayesian networks must be acyclic, and so set $P(S) = 0$ for any S containing cycles:

$$P(S) = \alpha C(S) \prod_{j=1}^{d'} P(s_j) \tag{6.4}$$

where $C(S) = 0$ if S contains a cycle and $C(S) = 1$ otherwise, and $P(s_j) = 1 - 2p_0$ if $S_j = \emptyset$, $P(s_j) = p_0$ if $S_j = \rightarrow$ or $S_j = \leftarrow$, and $P(s_j) = 0$ if $S_j = \leftrightarrow$. Combining Equations 6.3 and 6.4 yields

$$P(S|E) = \alpha P(S)P(E|S) = \alpha C(S) \prod_{j=1}^{d'} P(s_j) \prod_{i=1}^{m} P(e_{ij}|s_j) \tag{6.5}$$

6.1.3 Analysis

We now derive an expression for the probability that our method incorrectly predicts the value of a structure variable s_j, assuming that the expert model and parameters it uses are correct. Viewed another way, this is the expected fraction of incorrect arcs in the structure predicted by the ensemble of experts. More precisely, if $\hat{s}_j$ is the value of s_j with highest $p_j = P(s_j) \prod_{i=1}^{m} P(e_{ij}|s_j)$ (see Equation 6.5), the expression is for the probability that $\hat{s}_j \neq s_j$ (i.e., for the probability of making an incorrect prediction before breaking cycles). $P(\hat{s}_j \neq s_j)$ can be expanded as follows:

$$P(\hat{s}_j \neq s_j) = \sum_{v_j \in \{\emptyset, \rightarrow, \leftarrow, \leftrightarrow\}} P(s_j = v_j) P(\hat{s}_j \neq s_j | s_j = v_j) \tag{6.6}$$

In turn, $P(\hat{s}_j \neq s_j | s_j = v_j)$ can be expanded thus:

$$P(\hat{s}_j \neq s_j | s_j = v_j) = \sum_{\mathcal{E}} P(\hat{s}_j \neq s_j | e_{1:m,j}, s_j = v_j) \prod_{i=1}^{m} P(e_{ij} | s_j = v_j) \tag{6.7}$$

where the sum is over the set $\mathcal{E}$ of all 4^m possible vectors of expert statements $e_{1:m,j} = (e_{1,j}, \ldots, e_{mj})$ about s_j. The probabilities $P(e_{ij}|s_j = v_j)$ can be obtained from Table 6.1. $P(\hat{s}_j \neq s_j | e_{1:m,j}, s_j = v_j) = 0$ if p_j is higher for v_j than for any other value (i.e., the correct prediction is made), and

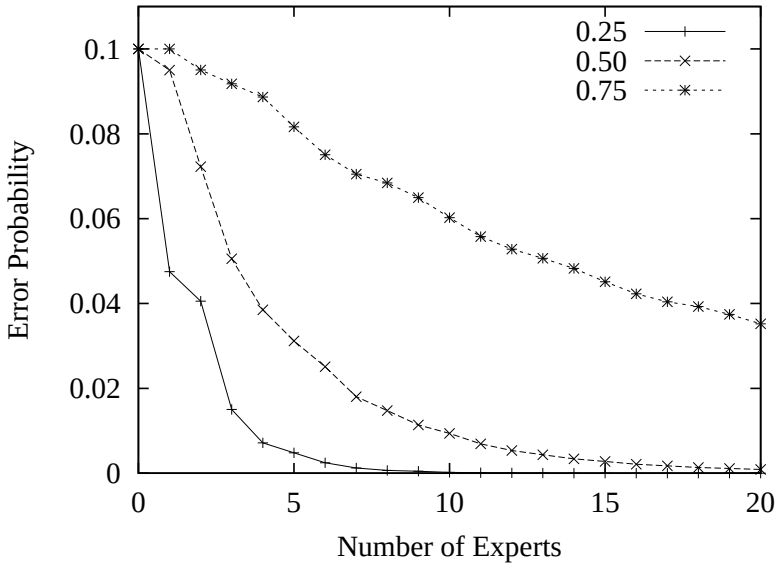


Figure 6.2: Error probability as a function of the number of experts and their noise level. Note that $p_0 = 0.05$, so with no experts, the "best guess" is no edge between the pair of nodes, leading to an error probability of $2p_0 = 0.1$ regardless of noise level.

$P(\hat{s}_j \neq s_j | e_{1:m,j}, s_j = v_j) = 1$ otherwise. Replacing this into Equation 6.7 and the latter into Equation 6.6, we obtain a long but straightforward expression for the error probability $P(\hat{s}_j \neq s_j)$. (The number of terms in Equation 6.7 can be greatly reduced by counting and combining all terms with the same number of experts predicting each value of s_j.) This probability as a function of the number of experts and the noise level is plotted in Figure 6.2. A noise level of ρ is defined as the expert parameter settings in which the expert removes or reverses a fraction ρ of the arcs, and adds the same number between unrelated nodes. We expect edge deletion to be much more frequent in practice than edge reversal, so we let $p_d = 4p_r$. We set p_0 (the probability that a pair of nodes is connected in a given direction) to 0.05. As can be seen, the error probability is low even with few experts and a substantial noise level. See also the analyses of model ensembles in Hansen & Salamon [59] and Perrone & Cooper [105].

6.1.4 Algorithm

We use the hill-climbing search algorithm of Heckerman et al. [61] to find the best structure, initializing it with the structure that maximizes $P(S|E)$, and replacing $P(S)$ by $P(S|E)$ in the BD score (Equation 2.1), as described in the previous section. However, learning the structure of a Bayesian network can be computationally expensive. A common alternative, if the expert knowledge is of sufficient quality, is to take the structure given by the expert(s) as the "true" one, and simply fit parameters to this structure. In our context, this means taking the structure with highest $P(S|E)$ as the "true" one, and learning parameters. We also explore this alternative in our experiments.

Consider Equation 6.5, and ignore for the moment the problem of cycles (i.e., the $C(S)$ term). Because the term for each structure variable s_j is independent of all others, the structure with highest $P(S|E)$ can be found in $O(d^2m)$ time simply by setting each s_j to the value with highest $p_j = P(s_j)\prod_{i=1}^{m} P(e_{ij}|s_j)$. If we assume that the structures provided by the experts have some sparse number of arcs $e \ll d^2$, and the expert parameters ($p_a, p_d, \ldots$) are set reasonably so that an arc can only be introduced between a pair of nodes j if $\exists i\ s_{ij} \neq \emptyset$, then this computation takes only $O(em)$ time. However, the structure S' found in this way may contain cycles. We thus heuristically find the structure with highest $P(S|E)$ by breaking all cycles in S' while minimally reducing its score $\prod p_j$. We use the procedure of Hulten el al. [64], which finds the set of arcs involved in cycles by finding the graph's strongly connected components, and breaks cycles by greedily removing the component arcs with lowest p_j. Though not guaranteed to find the acyclic structure with highest probability, in our experience this quickly finds an acyclic structure using very few arc removals.

We set the parameters p_a, p_d, p_r, p_b, p_c to optimize log-likelihood, measured by two-fold cross-validation. Optimization is done either by sequentially trying a pre-determined set of values for each parameter, or by Powell's method [107].

A further issue that arises is that the available data may be insufficient to reliably fit the parameters of the structure with highest $P(S|E)$, leading to poor performance, even if this structure is the "correct" one. In this case an oversimplified structure (with fewer parents per node, and thus more data for each CPT entry) might perform better. We address this issue by performing *shrinkage* of the parameters as follows. For each node x_i, we order the parents $par(x_i)$ as follows: $par_1(x_i)$ is the parent with highest mutual information with respect to x_i; $par_h(x_i)$ is the

parent with highest mutual information given $\{par_1(x_i), \ldots, par_{h-1}(x_i)\}$. Let $\hat{p}_{ijk0}$ be the unshrunken estimate of $p_{ijk} = P(x_i = k \mid par(x_i) = j)$ (i.e., conditioning on all parents). Let $\hat{p}_{ijks}$ be the estimate obtained by ignoring the last s parents in the ordering. The shrunken estimate is then $\hat{p}_{ijk} = \sum_{s=0}^{|par(x_i)|} \lambda_s \hat{p}_{ijks}$. The shrinkage coefficients λ are found using the EM algorithm, as described in McCallum et al. [87]. The shrinkage is incorporated into the parameter estimation and structure search by appropriately setting the Dirichlet parameters (see Equation 2.1): $n'_{ijk} = (n_{ij}/\lambda_0) \sum_{s=1}^{|par(x_i)|} \lambda_s p_{ijks} + 1$.

6.2 Experiments

We performed two sets of experiments. In the first, we used simulated experts and data generated from benchmark Bayesian networks to study the effect of training set size, number of experts, and noise level on our algorithm. In the second, we used knowledge of printer troubleshooting from nine computer users, and data from the Microsoft Windows printer diagnosis network.

6.2.1 Simulated Experts

We used four networks from the Bayesian network repository at http://www.cs.huji.ac.il/labs/compbio/Repository/ as our ground truths. Table 6.2 lists the networks and some of their characteristics. Since the correct prior probability of an edge, p_0^*, is unknown in practice, we set p_0 to be 0.05 for all networks. Expert parameters were set as in Section 6.1.3. Unless specified, the default noise level is 0.5 ($p_a = 0.025$, $p_d = 0.4$, $p_r = 0.1$, $p_c = 0$, $p_b = 0$), and the training set size is 100 examples.

We generated expert statements from the true networks according to the model described in the previous section. The parameters for the structure inference algorithm were not set to the true values, but optimized as they would need to be in a real-world situation.[2] After optionally performing a search over structures (guided by $P(S|E, D)$), the resulting network was evaluated using two measures: a) the average K-L distance from the true network, estimated using 100k samples,[3] and b) the structural difference between the learned network and the true one, measured as the number of arcs added or removed, with reversals counting as two differences.

We compared expert combination with three other cases: *zero experts* (purely empirical learning, starting from an empty network), *one expert* (the network provided by one expert), and *true* (purely

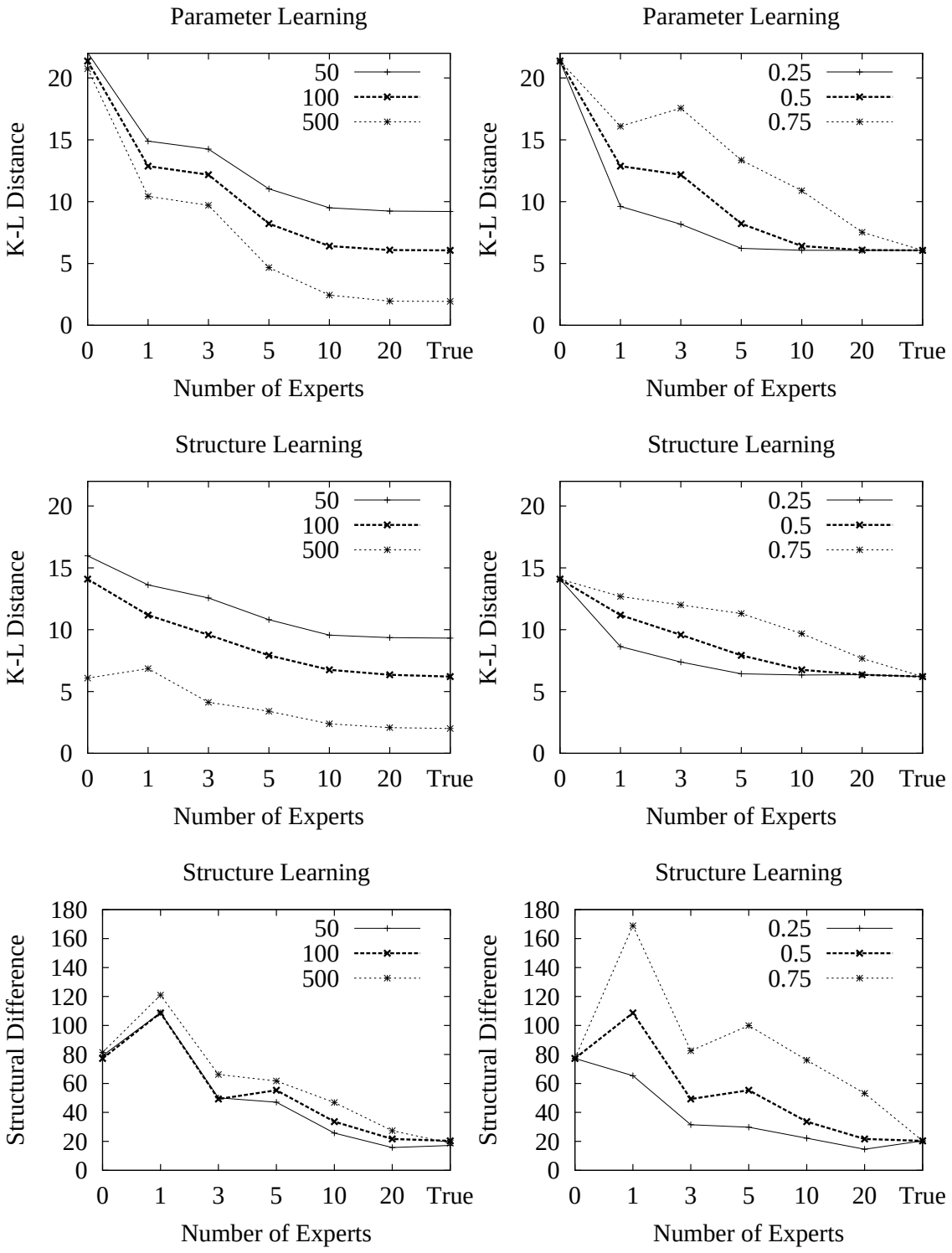


Figure 6.3: Experimental results for simulated experts: varying training set size (left) and varying noise level (right).

empirical learning, starting from the true network). For these we used the same prior $P(S)$ as Heckerman et al. [61], which is a discount of a factor of κ for every structural change from the initial model. We searched for the best κ in $\{0.01, 0.1, 0.2, \ldots, 0.8, 0.9, 0.99\}$ using two-fold cross-validation.[4] Results are averaged over 20 runs; each has independent training sets and experts, but all share the test set. Results for each run were on varying subsets of a fixed set of 20 experts. Within each run, each successively larger set of experts was obtained by adding randomly-chosen experts to the previous (smaller) set.

Results were qualitatively similar in all four domains. Table 6.2 shows summary results for parameter learning in the four domains. A single expert is useful, but still quite error-prone. In contrast, ten experts are sufficient to essentially recover the true network. Full results for hailfinder, the most complex domain, are shown in Figure 6.3.[5] The top graphs were obtained by finding the best structure using our method with 3 to 20 experts, and estimating parameters for it. "0" is the empty structure, "1" is a single expert's structure (with cycles removed), and "True" is the true structure. The middle and bottom graphs were obtained by finding the initial structure and prior over structures using our method with 3 to 20 experts, and applying Heckerman et al.'s (1995) structure-learning algorithm. In "0" the initial structure is empty, in "1" it is a single expert's structure, and in "True" it is the true structure; in these three cases, Heckerman et al.'s algorithm was applied with their prior. In all cases, network parameters were learned with shrinkage.

With or without structure learning, multiple experts systematically outperform a single expert, as well as purely empirical learning, in both K-L distance and structural difference. This illustrates the potential of our approach. A single expert outperforms structure learning in K-L distance but not in structural difference, suggesting that using multiple experts is particularly important when the goal is to understand the structure of the domain, rather than just obtain accurate predictions.

6.2.2 Real Experts

For real experts, we used the same domain as in Chapter 4: the Microsoft printer troubleshooting Bayesian network. Note though that rather than asking experts for rules, this time we simply asked for structural information. The network has 76 nodes and 112 edges, and $p_0^* = 0.02$.

We used the same experimental procedure as in the previous section. Our "experts" were nine

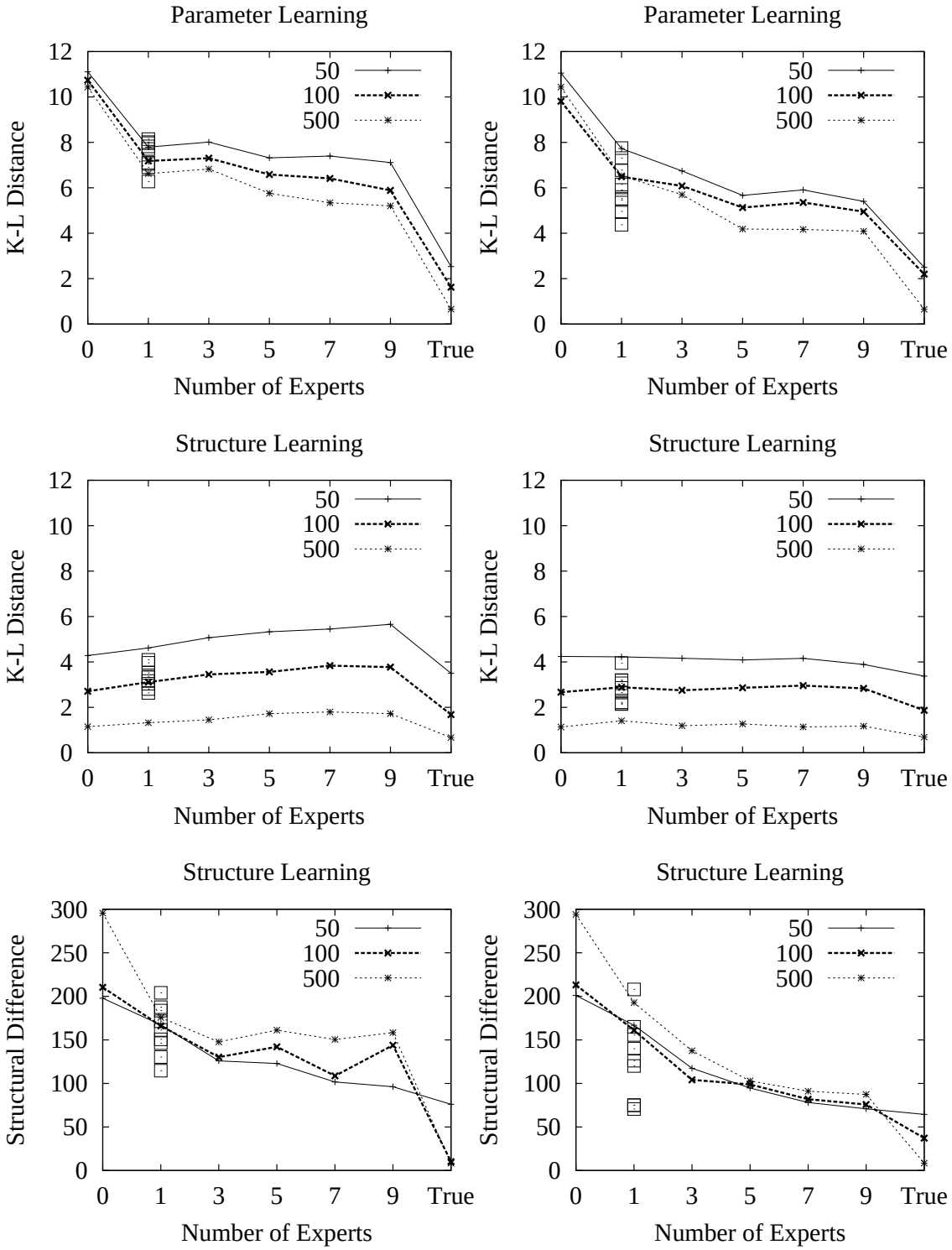


Figure 6.4: Experimental results in printer domain: low expertise (left) and high expertise (right).

Table 6.2: Network characteristics and results. p_0^* is the true probability of an arc between two nodes. δ_i is the reduction in K-L distance achieved when using i experts, as a fraction of the maximum possible (difference between learning with the empty network and learning with the the true one).

Network	Nodes	Edges	p_0^*	δ_1	δ_{10}
Alarm	37	46	0.035	47%	93%
Hailfinder	56	66	0.021	55%	98%
Insurance	27	52	0.074	57%	98%
Water	32	66	0.067	51%	98%

computer users who were familiar with printing but not intimately knowledgeable about the details of Windows printing. Each expert was given a list of the domain variables and asked to list the direct causes of each. To simulate more knowledgeable experts, we let each expert then study the true structure of the network for a period of time, asking them to try to understand the reason why certain variables were causes of other variables (we assisted in this task by providing an explanation for each). The experts' instructions are in Appendix C. After this, the experts were asked again to list the causes of each variable. We call these two cases "low-expertise" and "high-expertise" respectively. The experts spent from two to five hours on the task. We set $p_0 = 0.02$, and tried optimizing the parameters of the expert model both globally and separately for each expert. The latter alternative (with Powell's method) performed slightly better, and is the one we report.

The results, averaged over 20 runs, are shown in Figure 6.4. In each run, the subset of experts used for each number of experts was randomly chosen. The boxes represent the scores of the individual experts with 100 examples, averaged across runs. Without structure learning, experts always produced better networks than the purely empirical approach, and multiple experts combined using our method outperformed using a single one, on average. With structure learning, all methods performed similarly in K-L distance. This can be seen to be due to the lack of room for improvement between purely empirical learning and starting with the true network, and can be attributed to the simplicity of the domain. We expect the benefits of our approach to be more visible on large, complex domains where the data is generated by the real world, instead of by a model network. Ex-

perts outperformed purely empirical learning, and multiple experts outperformed one, in structural difference, showing the potential advantages of our method for obtaining insight.

The average K-L distance of the boxes (in the left and middle plots) is the K-L distance that would result from combining expert predictions with a logarithmic pool [40] (i.e., from combining the models that would result from fitting parameters separately to each expert structure). Without structure learning, the combined experts outperformed the logarithmic pool, and all but the best of the individual experts. This shows that our method can be preferable to standard model combination techniques. With structure learning, the K-L distances of our method and the logarithmic pool were similar, for the reasons described above. Even in this case, our method may still be preferable, because it produces a single comprehensible structure, in contrast to the multiple structures maintained by *a posteriori* combination methods.

Even on these simple networks, structure learning was an order of magnitude slower than parameter learning. In domains with many thousands of variables, where expertise combination is likely to be most useful, structure learning may simply not be feasible. Our method with only parameter learning is likely to be the best choice in this case. Given that structure learning is likely to be faster when starting closer to the true network, there may also be cases where our method with structure learning is feasible, while purely empirical structure learning is not.

In this and previous chapters, we have assumed there is some amount of training data available, either for learning parameters of the contributed knowledge, or for learning parameters about the experts themselves. What can we do if we have no such data? In the next chapter, we tackle this problem by using a *web of trust* to infer the quality of users based on how much they are trusted by other users known to the system.

Chapter 7

USING TRUST PROPAGATION TO WEIGHT CONTRIBUTIONS

In previous chapters, the purpose was to combine knowledge, particularly in the form of logical assertions, using inference and learning. Learning, in particular, was used to update the probabilistic parameters of the knowledge contained in the collective knowledge base. A problem thus arises if we have little or no training data with which to learn these parameters. In such a situation, how can the CKB determine the quality of the knowledge it has been given?

We tackle this problem by employing a web of trust, in which each user maintains trusts in a small number of other users. We then compose these trusts into trust values for all other users. We define properties for combination functions which merge such trusts, and define a class of functions for which merging may be done locally while maintaining these properties. We give examples of specific functions and apply them to data from Epinions (www.epinions.com), and our BibServ bibliography server. Experiments confirm that the methods are robust to noise, and do not put unreasonable expectations on users.

In the next section, we formulate a model that explicitly has the dual notions of *trust* and *belief*. We then define the meaning of belief combination under two different interpretations, and show an equivalence between the two. We also show a correspondence between combining beliefs and trusts that allows the use of whichever is more computationally efficient for the given system. We then give experimental results that show that our methods work across a wide variation of user quality and noise levels

7.1 Model

It is important to note that our focus here is not on deriving beliefs for new statements given an initial set of statements. Rather, we propose a solution to the problem of establishing the degree of belief in a statement that is explicitly asserted by one or more sources in our collective knowledge base.

These beliefs can then be used by an appropriate calculus to compute beliefs in derived statements.

Our basic model is that a user's belief in a statement should be a function of her trust in the sources providing it. Given each source's belief in the statement and the user's trust in each source, the user's belief in the statement can be computed in many different ways, corresponding to different models of how people form their beliefs. The framework presented in this chapter supports a wide variety of combination functions, such as linear pool [40][46], noisy OR [102], and logistic regression [4]. We view the coefficients in these functions (one per source) as measuring the user's trust in each source,[1] and answer the question: how can a user decide how much to trust a source she does not know directly? Our answer is based on recursively propagating trust: if A has trust u in B and B has trust v in C, then A should have some trust t in C that is a function of u and v. We place restrictions on allowable methods for combining trusts that enable the efficient and local computation of derived trusts. Similar restrictions on belief combination allow it to also be done using only local information.[2]

Consider a system of N users who, as a whole, have made M statements. Since we consider statements independently, we introduce the system as if there is only one.

Beliefs. Any user may assert her *personal belief* in the statement, which is taken from [0,1]. A high value means that the statement is accurate, credible, and/or relevant. Let b_i represent user i's personal belief in the statement. If user i has not provided one, we set b_i to 0. We refer to the collection of personal beliefs in the statement as the column vector **b** (using more complex beliefs and trusts is part of our plans for future work).

Trusts. User i may specify a *personal trust*, t_{ij}, for any user j. Trust is also a value taken from [0,1], where a high value means that the user is credible, trustworthy, and/or shares similar interests. If unspecified, we set t_{ij} to be 0. Note that t_{ij} need not equal t_{ji}. The collection of personal trusts can be represented as a N×N matrix **T**. We write $\mathbf{t}_i$ to represent the row vector of user i's personal trusts in other users. In the future, we would like to investigate more complex (e.g., topic-specific) trusts (see Section 8.2.4).

Merging. The web of trust provides a structure on which we may compute, for any user, their belief in the statement. We will refer these as *merged beliefs* ($\mathcal{B}$), to distinguish them from the user-specified *personal beliefs* (**b**). The trust between any two users is given by the *merged trusts matrix* ($\mathcal{T}$), as opposed to the user-specified *personal trusts matrix* (**T**).

7.2 Path Algebra Interpretation

In order to compute merged beliefs efficiently, we first make the simplifying assumption that a merged belief depends only on the paths of trust between the user and any other user with a personal belief in the statement. In Section 7.3 we consider an alternative probabilistic interpretation. For the moment, we consider only acyclic graphs (we generalize later to cyclic graphs).

Borrowing from generalized transitive closure literature [2], we define merged beliefs under the path algebra interpretation with the following conceptual computation (see Figure 7.1):

1. Enumerate all (possibly exponential number of) paths between the user and every user with a personal belief in the statement.

2. Calculate the belief associated with each path by applying a *concatenation function* to the trusts along the path and also the personal belief held by the final node.

3. Combine those beliefs with an *aggregation function*.

Some possible concatenation functions are multiplication and minimum value. Some possible aggregations functions are addition and maximum value. Various combinations lead to plausible belief-merging calculations such as measuring the most-reliable path or the maximum flow between the user and the statement.

Let $\circ$ and $\diamondsuit$ represent the concatenation and aggregation functions respectively. For example, $t_{ik} \circ t_{kj}$ is the amount that user i trusts user j *via k*, and the amount that i trusts j via any single other node is $\diamondsuit(\forall k : t_{ik} \circ t_{kj})$. If $\diamondsuit$ is addition and $\circ$ is multiplication, then $\diamondsuit(\forall k : t_{ik} \circ t_{kj}) \equiv \sum_k t_{ik} t_{kj}$. We define the matrix operation **C**=**A**•**B** such that $C_{ij} = \diamondsuit(\forall k : A_{ik} \circ B_{kj})$. Note that for the previous example, **A**•**B** is simply matrix multiplication.

7.2.1 Local Belief Merging

The global meaning of beliefs given above assumes a user has full knowledge of the network including the personal trusts between all users, which is unreasonable in practice. Can we instead merge beliefs locally while keeping the same global interpretation? Following Agrawal et al. [2],

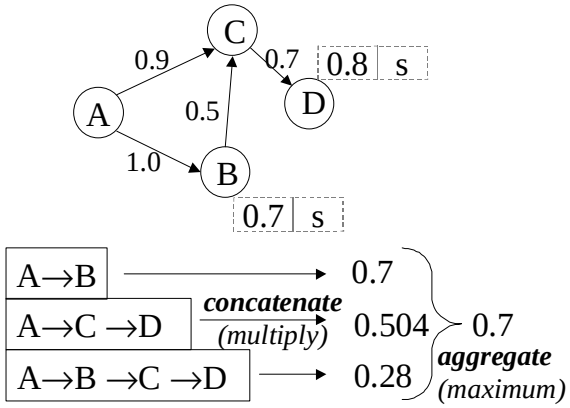


Figure 7.1: Path Algebra belief merging on an example web of trust.

let *well-formed decomposable path problems* be defined as those for which $\diamondsuit$ is commutative and associative, and $\circ$ is associative and distributes over $\diamondsuit$ (The above examples for $\diamondsuit$ and $\circ$ all result in well-formed path problems). These may be computed using *generalized transitive closure* algorithms, which use only local information. One such algorithm is as follows:

1. $\mathcal{B}^{(0)} = \mathbf{b}$

2. $\mathcal{B}^{(n)} = \mathbf{T} \bullet \mathcal{B}^{(n-1)}$, or alternatively, $\mathcal{B}_i^{(n)} = \diamondsuit(\forall k : t_{ik} \circ \mathcal{B}_k^{(n-1)})$

3. Repeat step 2 until $\mathcal{B}^{(n)} = \mathcal{B}^{(n-1)}$

(where $\mathcal{B}^{(i)}$ represents the value of $\mathcal{B}$ in iteration i. Recall $\mathcal{B}$ are the merged beliefs)

Notice that in step 2, the user needs only the merged beliefs of her immediate neighbors, which allows her to merge beliefs *locally* while keeping the same *global* interpretation. We will use the term *belief combination function* to refer to the above algorithm and some selection of $\circ$ and $\diamondsuit$.

7.2.2 Strong and Weak Invariance

Refer to Figure 7.2 (*Case I*). Suppose a node is removed from the web of trust, and the edges to it are redirected to its trusted nodes (combining the trusts). If the merged beliefs of the remaining

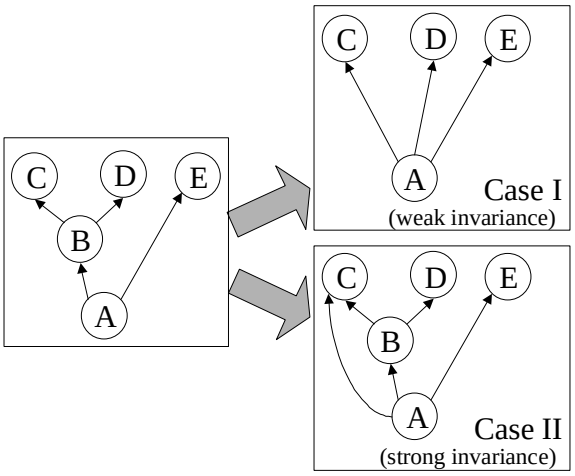


Figure 7.2: Strong and weak invariance.

users remain unchanged, we say the belief combination function has *weak global invariance*. The path interpretation has this important property.

We can imagine another property that may be desirable. Again refer to Figure 7.2 (*Case II*). If we add an arc of trust directly from A to C, and the trust between A and C is unchanged, we say that the belief combination function has *strong global invariance.* Any belief combination function with weak invariance for which the aggregation function is also *idempotent* (meaning, $\diamondsuit(x, x) = \diamondsuit(x)$) will have strong invariance. This follows from the fact that the aggregation function is associative. Interestingly, whether or not the aggregation function must be idempotent is the primary difference between Agrawal et al.'s well-formed decomposable path problems [2] and Carre's path algebra [15] (also related is the definition of a closed semiring in [5]). One example of a belief combination function with strong global invariance is the one defined with $\diamondsuit$ as maximum and $\circ$ as multiplication.

7.2.3 Merging Trusts

The majority of the belief merging calculation involves the concatenation of chains of trust. Beliefs only enter the computation at the endpoint of each path. Instead of merging beliefs, can we merge trusts and then reuse these to calculate merged beliefs?

We define the interpretation of globally merged trusts in the same way as was done for beliefs:

the trust between user i and user j is an aggregation function applied to the concatenation of trust along every path between them. It falls directly from path algebra that, if $\diamondsuit$ is commutative and associative, and $\circ$ is associative and distributes over $\diamondsuit$, then we can combine trusts locally while still maintaining global meaning:

1. $\mathcal{T}^{(0)} = \mathbf{T}$
2. $\mathcal{T}^{(n)} = \mathbf{T} \bullet \mathcal{T}^{(n-1)}$
3. Repeat step 2 until $\mathcal{T}^{(n)} = \mathcal{T}^{(n-1)}$

($\mathcal{T}^{(i)}$ is the value of $\mathcal{T}$ in iteration i. Recall that $\mathcal{T}$ is the matrix of merged trusts). To perform the computation, a user needs only to know her neighbors' merged trusts. This leads us to the following theorem, which states that, for a wide class of functions, merging trusts accomplishes the same as merging beliefs (the proof is in Appendix D)

Theorem 7.2.1 *If $\diamondsuit$ is commutative and associative, and $\circ$ is associative and distributes over $\diamondsuit$, and $\boldsymbol{T}$, $\mathcal{T}$, $\boldsymbol{b}$, and $\mathcal{B}$ are as above, then $\boldsymbol{T} \bullet \mathcal{B} = \mathcal{T} \bullet \boldsymbol{b}$.*

7.2.4 Cycles

Thus far, we have assumed the graph is acyclic. However, it is improbable that a web of trust will be acyclic. Indeed, the Epinions web of trust (see Section 7.5) is highly connected and cyclic. Borrowing terminology from path algebra, we define a combination function as *cycle-indifferent* if it is not affected by introducing a cycle in the path between two users. With cycle indifference, the aggregation over infinite paths will converge, since only the (finite number of) paths without cycles affect its calculation.

Proposition 7.2.2 *Theorem 7.2.1 is applicable to cyclic graphs if $\diamondsuit$ and $\circ$ define a cycle-indifferent path problem.*

On cyclic graphs, a combination function that is not cycle-indifferent has the questionable property that a user may be able to affect others' trusts in him by modifying her own personal trusts.

However, requiring a cycle-indifferent combination function may be overly restrictive. In Section 7.3 we explore an alternative interpretation that allows the use of combination functions that are not cycle-indifferent.

7.2.5 Selection of Belief Combination Function

The selection of belief combination function may depend on the application domain, desired belief and cycle semantics, and the expected typical social behavior in that domain. The ideal combination function may be user-dependent. For the remainder of the chapter, we will always use multiplication for concatenation, though in the future we would like to explore other functions (such as the minimum value). The following is a brief summary of three different aggregation functions we have considered.

- **Maximum Value.** Using maximum to combine beliefs is consistent with fuzzy logic, in which it has been shown to be the most reasonable function for performing a generalized *or* operation over [0,1] valued beliefs [9]. Maximum also has the advantages that it is cycle-indifferent, strongly consistent, and naturally handles missing values (by letting them be 0). With maximum, the user will believe anything believed by at least one of the users she trusts – a reasonable, if not overly optimistic, behavior.

- **Minimum Value.** Minimum is not cycle-indifferent. In fuzzy logic, minimum value is used to perform the *and* operation. With minimum, the user will only believe a statement if it is believed by all of the users she trusts.

- **Average.** Average does not satisfy the requirements for a well-formed path algebra outlined above (average is not associative). However, average can still be computed by using two aggregation functions: sum and count (count simply returns the number of paths by summing 1's). By passing along these two values, each node can locally compute averages. Average is not cycle-indifferent.

7.2.6 Computation

Since cycle-indifferent, weakly consistent combination functions yield well-formed path problems, $\mathcal{B}$ and $\mathcal{T}$ may be computed using standard transitive closure algorithms. The simplest of these is the semi-naïve algorithm [8], which runs in O(N^4) time, and essentially prescribes repeated application of the belief update equation. If running as a peer-to-peer system, the semi-naïve algorithm may be easily parallelized, requiring O(N^3) computations per node [3]. Another algorithm is the Warshall algorithm [124], which computes the transitive closure in O(N^3). Some work on parallel versions of the Warshall algorithm has been done in Agrawal & Jagadish [3]. There has also been much research on optimizing transitive closure algorithms, such as for when the graph does not fit into memory [2]. In practice most users will specify only a few of the users as neighbors, and the number of iterations required to fully propagate information is much less than N, making the computation quite efficient. Theorem 7.2.1 allows us to choose whether we wish to merge trusts or merge beliefs. The most efficient method depends on, among other things, whether the system is implemented as a peer-to-peer network or as a server, the number of neighbors for a given user, the number of users, the number of statements in the system, and the number of queries made by each user.

7.3 Probabilistic Interpretation

In this formulation, we consider a probabilistic interpretation of global belief combination. The treatment is motivated by random walks on a Markov chain, which have been found to be of practical use in discovering high-quality web pages [99]. In what follows, we assume the set of personal trusts for a given user has been normalized (i.e., it sums to one).

Imagine a random knowledge-surfer hopping from user to user in search of beliefs. At each step, the surfer probabilistically selects a neighbor to jump to according to the current user's distribution of trusts. Then, with probability equal to the current user's belief, it says "yes, I believe in the statement". Otherwise, it says "no". Further, when choosing which user to jump to, the random surfer will, with probability $\lambda_i \in [0, 1]$, ignore the trusts and instead jump directly back to the original user, i. We define a combination method to have a *global probabilistic interpretation* if it satisfies the following:

1. $\mathcal{T}_{ij}$ is the probability that, at any given step, user i's random surfer is at user j.

2. $\mathcal{B}_i$ is the probability that, at any given step, user i's random surfer says "yes".

The convergence properties of such random walks are well studied; $\mathcal{B}$ and $\mathcal{T}$ will converge as long as the network is irreducible and aperiodic [89]. λ_i can be viewed as a *self-trust*, and specifies the weight a user gives to her own beliefs and trusts. The behavior of the random knowledge-surfer is very similar to that of the intelligent surfer presented in Richardson & Domingos [113], which is a generalization of PageRank that allows non-uniform transitions between web pages. What personalizes the calculation to user i is the random restart, which "grounds" the surfer to i's trusts. The resulting trusts may be very different than using PageRank.

7.3.1 Computation

User i's trust in user j is the probability that her random surfer is on a user k, times the probability that the surfer would transition to user j, summed over all k. Taking λ_i into account as well, we have

$$\mathcal{T}_{ij} = \lambda_i \delta(i-j) + (1-\lambda_i) \sum\nolimits_k \mathcal{T}_{ik} t_{kj} \tag{7.1}$$

where $\delta(0) = 1$ and $\delta(x \neq 0) = 0$ and each row of $\mathbf{T}$ is normalized. In matrix form:

$$\mathcal{T}_i = \lambda_i \mathbf{I}_i + (1-\lambda_i)\, \mathcal{T}_i \mathcal{T}, \tag{7.2}$$

where $\mathbf{I}_i$ is the i^{th} row of the identify matrix. In order to satisfy the global probabilistic interpretation, $\mathcal{B}_i$ must be the probability that user i's random surfer says "yes". This is the probability that it is on a given user times that user's belief in the statement:

$$\mathcal{B}_i = \sum\nolimits_k \mathcal{T}_{ik} b_k, \text{ or, } \mathcal{B}_i = \mathcal{T}_i \mathbf{b} \tag{7.3}$$

7.3.2 Local Belief and Trust Merging

As in section 7.2.1, we wish to perform this computation using only local information. We show that this is possible in the special case where $\lambda_i = \lambda$ is constant. Unrolling Equation 7.2:

$$\mathcal{T} = \lambda \left[\sum_{m=0}^{\infty} (1-\lambda)^m \mathbf{T}^m \right]. \tag{7.4}$$

Note that $\mathcal{T}^0 = \mathbf{I}$. Substituting into Equation 7.3,

$$\mathcal{B} = \lambda \left[\sum_{m=0}^{\infty} (1-\lambda)^m \mathbf{T}^m \right] \mathbf{b}, \tag{7.5}$$

which is satisfied by the recursive definition:

$$\mathcal{B} = \lambda \mathbf{b} + (1-\lambda)\, \mathbf{T}\mathcal{B} \tag{7.6}$$

Thus we find that in order to compute her merged belief, each user needs only to know her personal belief, and the merged beliefs of her neighbors. Besides having intuitive appeal, this has a probabilistic interpretation as well: user i selects a neighbor probabilistically according to her distribution of trust, $\mathbf{t}_i$, and then, with probability $(1-\lambda)$, accepts that neighbor's (merged) belief, and with probability λ accepts her own belief. Further, Equation 7.4 is also equivalent to the following, which says that a user may compute her merged trusts knowing only the merged trusts of her neighbors:

$$\mathcal{T} = \lambda \mathbf{I} + (1-\lambda)\, \mathbf{T}\mathcal{T} \tag{7.7}$$

The probabilistic interpretation for belief combination is essentially taking the weighted average of the neighbors' beliefs. We will thus refer to this belief combination as *weighted average* for the remainder of the chapter. Note that for weighted average to make sense, if the user has not specified a belief we need to impute the value. Techniques such as those used in collaborative filtering [112] and Bayesian networks [17] for dealing with missing values may be applicable. If only relative rankings of beliefs are necessary, then it may be sufficient to use 0 for all unspecified beliefs.

7.4 Similarity of Probabilistic and Path Interpretations

There are clearly many similarities between the probabilistic and path interpretations. In both, beliefs may be merged by querying neighbors for their beliefs, multiplying (or concatenating) those by the trust in each neighbor, and adding (or aggregating) them together. Both interpretations also allow the computation of merged beliefs by first merging trusts. If we let the aggregation function be addition, and the concatenation function be multiplication, then the only difference between the two interpretations is due to the factor, λ. If $\lambda = 0$, then Equation 7.6 for computing $\mathcal{B}$ is functionally the same as the algorithm for computing $\mathcal{B}$ in the path algebra interpretation. However, consider

this: If λ is 0 then Equation 7.2 for computing $\mathcal{T}_i$ simply finds the primary eigenvector of the matrix **T**. Since there is only one primary eigenvector, this means that $\mathcal{T}_i$ would be the same for all users (assuming the graph is aperiodic and irreducible). How do we reconcile this with the path algebra interpretation, in which we expect different trust vectors per user? The answer is that the corresponding path algebra combination function is not cycle indifferent, and as a result the user's personal beliefs will get "washed out" by the infinite aggregation of other users' beliefs. Hence, as in the probabilistic interpretation, all users would end up with the same merged beliefs.

Both methods share similar tradeoffs with regards to architectural design. They may easily be employed in either a peer-to-peer or client-server architecture. We expect the system to be robust because a malicious user will be trusted less over time. Further, since the default trust in a user is 0, it is not useful for a user to create multiple pseudonyms, and users are motivated to maintain quality of information.

The web of trust calculation is not susceptible to "link-spamming," a phenomenon in PageRank whereby a person may increase others' trust in him by generating hundreds of virtual personas which all trust him. In PageRank, the uniform random jump of the surfer means that each virtual persona is bestowed some small amount of PageRank, which they 'give' to the spammer, thus increasing her rank. With a web of trust, this technique gains nothing unless the user is able to convince others to trust her virtual personas, which we expect will only occur if the personas actually provide useful information.

7.5 Experiments

In this section, we measure some properties of belief combination using the methods presented above. We present two sets of experiments. The first uses a real web of trust, obtained from Epinions (www.epinions.com), but uses synthetic values for personal beliefs and trusts. We wanted to see how *maximum* (path interpretation) compared with *weighted average* (probabilistic interpretation) for belief combination. We also wanted to see what quality of user population is necessary for the system to work well, and what happens if there is a mix of low and high quality users. Finally, these methods would have little practical use if we required that users be perfect at estimating trusts of their neighbors, so we examine the effect that varying the quality of trust estimation has

on the overall accuracy of the system. For the second experiment, we implemented a real-world application, now available over the web (BibServ, www.bibserv.org). BibServ provides us with both anecdotal evidence and experimental results.

7.5.1 Experiments with the Epinions Web of Trust

For these experiments, we used the web of trust obtained from Epinions, a user-oriented product review website. In order to maintain quality, Epinions encourages users to specify which other users they trust, and uses the resulting web of trust to order the product reviews seen by each person[3]. In order to perform experiments, we needed to augment the web of trust with statements and real-valued trusts.

We expected the information from contributors to be of varying quality, so we assigned to each user i a *quality* $\gamma_i \in [0, 1]$. A user's quality determined the probability that a statement by the user was true. Unless otherwise specified, the quality of a user was chosen from a Gaussian distribution with $\mu = 0.5$ and $\sigma = 0.25$. These parameters are varied in the experiments below.

The Epinions web of trust is Boolean, but our methods require real-valued trusts. We expected that over time, the higher a user's quality, the more they were likely to be trusted. So, for any pair of users i and j where i trusts j in Epinions:

$$t_{ij} \sim U\left(\max\left(\gamma_j - \delta_{ij}, 0\right), \min\left(\gamma_j + \delta_{ij}, 1\right)\right) \tag{7.8}$$

$U(a, b)$ is the uniform distribution between a and b and 0 elsewhere, γ_i is the quality of user i and δ_{ij} is a noise parameter that determines how accurate users were at estimating the quality of the user they were trusting. We supposed that a user with low quality was bad at estimating trust, so for these experiments we let $\delta_{ij} = (1 - \gamma_i)$.

We generated a random world that consisted of 5000 true or false "facts" (half of the facts were false). Users' statements asserted the truth or falsity of each fact (there were thus 10,000 possible statements, 5000 of which were correct). A user's personal belief (b_i) in any statement she asserted was 1.0.

The number of statements made by a user was equal to the number of Epinions reviews that user wrote. The few users with highest connectivity tended to have written the most reviews, while the majority of users wrote few (or none).

Table 7.1: Average precision and recall for various belief combination functions, and their standard deviations.

Combination Function	**Precision**	**Recall**
Maximum	0.87± 0.13	0.98± 0.13
Weighted Average	0.69± 0.06	0.98± 0.15
Local	0.57± 0.13	0.44± 0.32
Random	0.51± 0.05	0.99± 0.11

For each fact, each user computed her belief that the fact was true and her belief that the fact was false. For each user i, let S_i be the set of statements for which $\mathcal{B}_i > \tau$. If a user had non-zero belief that a fact was true and a non-zero belief that a fact was false, we used the one with highest belief. Let G_i be the set of correct statements "reachable" by user i (a statement is reachable if there is a path in the web of trust from user i to at least one user who has made the statement). Then $S_i \cap G_i$ is the set of statements that user i correctly believed were true, so $precision_i = \frac{|S_i \cap G_i|}{|S_i|}$ and $recall_i = \frac{|S_i \cap G_i|}{|G_i|}$ Precision and recall could be traded off by varying the belief threshold, τ. We present precision and recall results averaged over all users, and at the highest recall by using τ=0. We now present results of four experiments:

- **Comparing Combining Functions**. In Table 7.1, we give results for a variety of belief combination functions. The combination functions *maximum* and *weighted average* are the same as introduced earlier (unless otherwise specified, λ is 0.5 for weighted average). With *random*, $\mathcal{T}_{ij}$ was chosen uniformly from [0,1]. Since the average quality is 0.5, half of the facts in the system are true, so *random* led to a precision of (roughly) 0.5. *Local* means that a user incorporated only the personal beliefs of her immediate neighbors, and resulted in a precision of 0.57. Weighted average and maximum significantly outperformed the baseline functions, and maximum outperformed weighted average. We found that (data not presented) the precision differed only slightly between users with high quality and users with low quality. We believe this is because a low quality user would still have good combined beliefs if all of her neighbors had good combined beliefs.

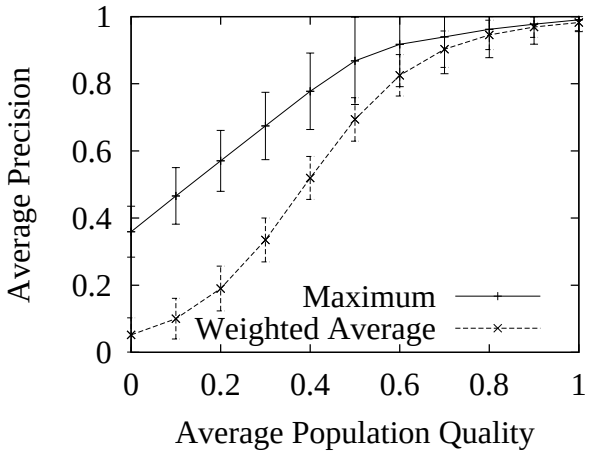


Figure 7.3: Average precision ($\pm\sigma$) for *maximum* and *weighted average*.

- **Varying the Population Quality.** It is important to understand how the average precision is affected by the quality of the users. We explored this by varying μ, the average population quality (see Figure 7.3). Overall, maximum significantly outperformed weighted average ($p < 0.01$), with the greatest difference at low quality. This may be due to the experimental set up, in which the accuracy of the trust depends on the quality of the user making the statement of trust. At low μ, there are few users with high quality, which will be well trusted on average, but the trust in them will be very noisy due to the overall low average quality. With maximum, each user needs just one path of high trust between herself and the good user. With weighted average, the few high trust paths are overwhelmed by the significantly larger number of low trust and simply noisy paths, resulting in too much uncertainty. Another way to look at it is that maximum filters out all noise by considering only the most trusted opinion, while weighted average incorporates all opinions. When the average quality of the users is high, the two should be similar, but when the average quality is low, maximum's ability to filter information allows it to produce consistently better results. Whether this is a phenomenon of the simulation or one which will be reflected in the real world remains to be seen.

  We also explored the effect of varying λ for *weighted average*. In Figure 7.4, we see that λ had only a small effect on the results. We found that the better the population, the lower

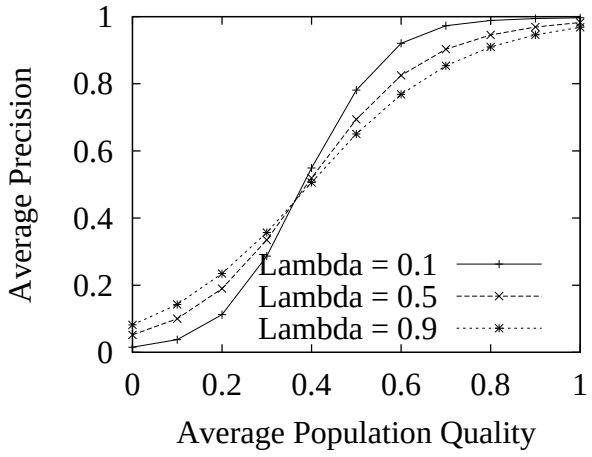


Figure 7.4: Effect of λ on the precision when combining with *weighted average*.

λ should be, which makes sense because in this case the user should put high trust in the population. Because maximum seemed to consistently outperform weighted average, and has the additional advantage of being cycle-indifferent and producing absolute beliefs, we restricted the remaining experiments to it.

- **Good and Bad Users.** To measure the robustness of the network to bad (or simply clueless) users, we selected user qualities from two Gaussian distributions, with means of 0.25 (*bad*) and 0.75 (*good*) (both had the same standard deviation as earlier, 0.25). We varied the fraction of users drawn from each distribution.

  We found the network to be surprisingly robust to bad users (see Figure 7.5). The average precision was very high (80-90%) when only 10-20% of the users were good. Consider also the network for which the fraction of good people is 0.5. This network has the same average population quality as the network used for Table 7.1, except in this case the population is drawn from a bimodal distribution of users instead of a unimodal distribution. The result is a higher precision, which shows that it is better to have a few good users than many mediocre ones.

- **Varying Trust Estimation Accuracy.** We also investigated how accurate the trusts must be

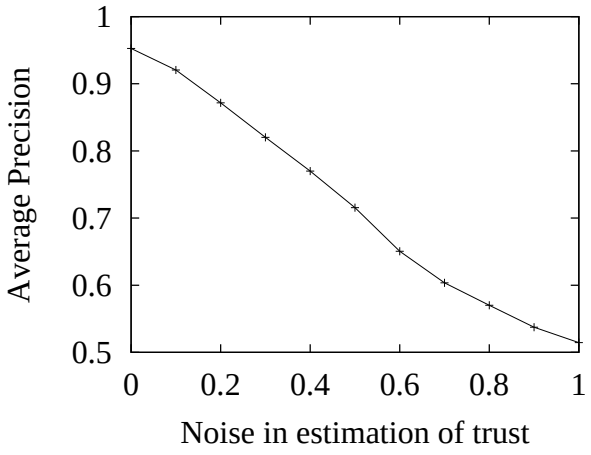


Figure 7.5: Precision for various fractions of good people in the network, using *maximum* belief combination.

in order to maintain good quality beliefs. We let the trust noise parameter be the same for all users ($\delta_{ij} = \delta$) and varied δ (see Equation 7.8). Note that when $\delta = 0$, t_{ij} was γ_j, and when $\delta = 1$, t_{ij} was chosen uniformly from [0,1]. Figure 7.6 shows the average precision for various values of δ. Even with a noise level of 0.3, acceptable precision (>80%) was maintained.

7.5.2 Experiments with the BibServ Bibliography Server

We have implemented our belief and trust combination methods in our BibServ system, which is publicly accessible at www.bibserv.org. BibServ is a bibliography service that allows users to search for bibliography entries for inclusion in technical publications. Users may upload and maintain their bibliographies, create new entries, use entries created by others, and rate and edit any entry.

Why Bibliographies?

We felt that bibliographies have many characteristics that make them a good starting point for our research into collective knowledge. The bibliography domain is simple, yet gives rise to all of the issues of information quality, relevance, inconsistency, and redundancy that we desire to research. The BibServ beta site currently has 127 users, drawn mainly from the UW computer science depart-

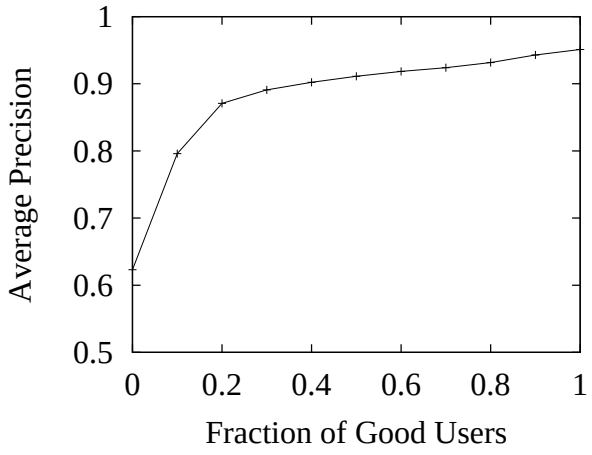


Figure 7.6: Effect of varying the quality of trust estimation.

ment and IBM Almaden, and half a million entries, of which 19250 were entered by the users.

Implementation.

BibServ is implemented as a centralized server, so we chose to store the merged trusts $\mathcal{T}$ and compute the merged beliefs as needed. This requires O(NM) space. Since there are many more bibliography entries than users, this is much less than the O(M^2) space that would be required if we instead stored the merged beliefs.

By our definition, a user's merged belief in a bibliography entry represents the quality and relevance of that entry to them. Hence, search results are ordered by belief.[4] The computation of merged trusts and beliefs is implemented in SQL and, in the case of beliefs, is incorporated directly into the search query itself. The overhead of computing beliefs is typically less than 10% of the time required to perform the query itself. Experiments were performed using weighted average ($\lambda = 0.5$) as well as maximum as belief combination functions.

Belief as Quality and Relevance.

The relation of belief combination to BibServ is as follows. When performing a search on BibServ, a user presumably is looking for a good bibliographic entry (e.g., one that has all of the important fields filled in correctly) that is related to her own field of study. Our concept of "belief" corresponds

to this – a good and relevant entry should have high belief. We treat each entry as a statement. Users may set their beliefs explicitly, and we implicitly assume a belief of 1.0 for any entry in their personal bibliography (unless otherwise explicitly rated). This forms the vector **b** for each entry. BibServ users are also presented with a list of other users whom they may rate. A high rating is intended to mean they expect the user to provide entries which are high quality and relevant. This forms the trust matrix **T**.

Experimental Results.

We asked BibServ users to think of a specific paper they were interested in, and use BibServ to search for it using keywords. We returned the search results in random order, and asked the user to rate each result for quality (0-5) and relevance (either "yes, this is the paper I was looking for" or "no, this is not"). We required the user to make the search general enough to return at least 5 entries, and to rate them all. We used two metrics to evaluate the results. The first is whether there was a correlation between beliefs and either the rated quality or relevance of the entries. In many cases, such as ordering search results, we only care whether the best k results may be determined by belief. We thus calculated the ratio of the average rating of the top k results (ordered by belief) vs. the average rating of all results. Unfortunately, we could do this experiment with only a small number of users. The data set consists of 405 ratings of quality and relevance on 26 searches by 13 users. The average user involved in the study specified 9 trusted users. Because the results are based on a small quantity of data, they should at best be considered preliminary.

The highest correlation was obtained with weighted average, which produced beliefs that had a correlation of 0.29 with the quality ratings ($\lambda = 0.03$). The other correlations were 0.10 (weighted average vs. relevance), 0.16 (maximum vs. quality), and -0.01 (maximum vs. relevance). These results are not as positive as we had hoped for. Many factors can contribute to a low correlation, such as having little variance in the actual quality and relevance of the entries. Currently, almost all of the entries in BibServ are related to computer science, and all of the users are computer scientists, so the web of trust gives little predictive power for relevance. We expect that as BibServ accumulates users and publications on more varying topics, the correlation results will improve.

The average ratio of the top k results to the rating of all results (across different searches) for

relevance ranged from 1.2 to 1.6 for a variety of k (1-5) and for either belief combination function. The average ratio ranged from 0.96 to 1.05 for quality. The ratio rapidly tended toward 1.0 as k increased, indicating that, while belief was a good indicator for relevance, the data contained a lot of noise (making it possible only to identify the very best few entries, not order them all). This is consistent with the low relevance correlation found above.

The most interesting result of these experiments was with regard to λ. We found that the best results when measuring beliefs vs. quality ratings were when λ was very small, though still non-zero. On the other hand, the best results for relevance were when λ was very large, though not equal to one. This indicated that 1) Most users shared a similar metric for evaluating the quality of a bibliography entry, and 2) Users had a widely varying metric for evaluating an entry's relevance. The best λ was not 0 or 1, indicating that both information from others and personalized beliefs were useful.

7.6 Related Work

The idea of a *web of trust* is not new. As mentioned, it is used by Epinions for ordering product reviews. Cryptography also makes use of a web of trust to verify identity [12]. In Abdul-Rahman's system, John's trust in Jane, and John's trust in Jane's ability to determine who is trustworthy, are separate, though discrete and only qualitatively valued [1]. Such a separation would be interesting to consider in our framework as well.

The analog of belief combination for the World-Wide Web is estimating the quality and relevance of Web pages. Information retrieval methods based solely on the content of the page (such as TFIDF [67]) are useful, but are outperformed by methods that also involve the connectivity between pages [16][75][99].

Gil and Ratnaker [52] present an algorithm that involves a more complex, though qualitative, form of trust based on user annotations of information sources, which are then combined. One shortcoming of such an approach is that it derives values of "trustworthiness" that are not personalized for the individual using them, requiring all users – regardless of personal values – to agree on the credibility of sources. Secondly, by averaging the statements of many users, the approach is open to a malicious attacker who may submit many high (or low) ratings for a source in order to hide

its true credibility. By employing a web of trust, our approach surmounts both of these difficulties (assuming users reduce their trust in a user that provides poor information).

Kamvar et. al's EigenTrust algorithm [68], which computes global trusts as a function of local trust values in a peer-to-peer network, is very similar to our probabilistic interpretation of trusts presented in section 7.3. One key difference is that we allow trusts and beliefs to vary; they are personalized for each user based on her personal trusts. In contrast, EigenTrust computes a global trust value (similar to PageRank) and emphasizes security against malicious peers who aim to disturb this calculation.

Social network algorithms have been applied to webs of trust in order to identify users with high network influence [33][114]. Applying the same methods to the CKB's web of trust may prove fruitful in identifying useful contributors, highly respected entities, etc. Also in a similar vein is the ReferralWeb project, which mines multiple sources to discover networks of trust among users [71]. Also interesting is collaborative filtering [112], in which a user's belief is computed from the beliefs of users she is similar to. This can be seen as forming the web of trust implicitly, based solely on similarity of interests.

Chapter 8

CONCLUSION

Knowledge acquisition is the key bottleneck preventing the wider spread of AI systems. Both current approaches to it — manual and automatic — have limitations that are hard to overcome. The Internet makes possible a new alternative: building knowledge bases by mass collaboration. While this approach can greatly reduce the time and cost of developing very large knowledge bases, it raises problems of quality, consistency, relevance, scalability and motivation. In this thesis, we examined these issues and proposed solutions to them. We developed an architecture which interacts with users and contributors to answer questions from users and update knowledge from contributors. We introduced Markov logic networks, a representation language that combines probability with first-order logic. We also demonstrated methods for combining structural information, and using a web of trust when there is a lack of training data. Throughout, we have demonstrated the utility of collective knowledge bases and the various methods through experiments in real-world domains.

In the next two sections, we review the contributions made by this thesis, and outline some directions for future work.

8.1 Contributions of this Thesis

Collective knowledge is a nascent field with many unsolved problems. This dissertation makes five significant contributions to the field, bringing the construction and use of CKBs closer to practical reality.

The first contribution is the development of an architecture for collective knowledge bases that addresses the problems of quality, consistency, relevance, scalability and motivation (Chapter 4). It uses a close interaction between users and contributors to keep the knowledge relevant. Inconsistency is handled by using probabilistic reasoning, and the quality of the knowledge is maintained by using machine learning based on user feedback. The algorithms employed scale well, and allow the

assignment of credit to useful contributors, thus enabling a variety of forms of motivation.

The second contribution is the development of Markov logic networks (MLNs), a novel representation language that combines the full power of first-order logic and probability in a simple framework (Chapter 5). The representation supersedes existing techniques, which are restricted to subsets of first-order logic (typically Horn clauses or description logic). We showed how to perform learning and inference with MLNs, and demonstrated their use with experiments. Due to their clean handling of noisy and inconsistent knowledge, Markov logic networks are a particularly appropriate representation for collective knowledge bases.

Our third contribution is a technique for combining structure information from multiple experts. Collective knowledge bases should be able to accept knowledge of various forms, one of which could be structural information (i.e dependencies between concepts). The ability to combine this information from multiple (potentially conflicting) experts is thus a useful development for collective knowledge bases.

The fourth contribution is a framework for inferring the quality of contributors by using a web of trust. In many domains, we will not have enough (or any) training data with which to determine the quality of the experts. In this case, a web of trust may be used. We presented two methods for computing trusts, which need only local information, and showed a correspondence between them.

The final contribution is a set of experiments that demonstrate the utility of collective knowledge bases and associated techniques. In the printer domain, we showed that combining the (very noisy) knowledge of four experts into a collective knowledge base outperformed any individual expert or pooling the experts. We also showed, in a synthetic domain, that the collective knowledge base performs significantly better than pooling the experts, and that the time required for inference grew sub-linearly in the size of the collective knowledge base. We found similar results for a collective knowledge base consisting of only the structure of a domain. Our experiments with Markov logic networks showed that they outperform propositional methods, or methods that use logical reasoning alone. The improvements obtained by Markov logic networks became even more significant when there was missing information.

Thus, we have promising evidence to conclude that collective knowledge bases, using Markov logic networks as their representation, can handle the issues of quality, consistency, and scalability. We believe they have significant potential for use on the Internet.

8.2 Future Work

Though much has been accomplished, there are also many directions for future work. In this section, we identify some of these, categorized by the contribution to which they apply.

8.2.1 Architecture

Although the architecture informs experts of the usefulness of their knowledge, we have not yet developed techniques for guiding contributors to where new knowledge would be most useful. By using value of information techniques [63], we may be able to find gaps in the knowledge base that have the highest (expected) utility when filled. Similarly, we would like to investigate the use of machine learning techniques to automatically propose refinements of knowledge. Such refinements could be suggested to the contributor who provided the knowledge, or alternatively, to other contributors. This could lead to multiple contributor "roles": knowledge creator, editor, etc.

In Chapter 4, we gave an algorithm for credit assignment. The algorithm is just one of many possible; studying these is one direction for future work.

In this work, we implicitly assumed that all contributors and users use the same vocabulary. This is unlikely to be true in a real system. We would like to automatically translate between the ontologies used by different contributors, possibly building on work such as that by Doan et al. ([30]) and Dou et al. ([34]).

8.2.2 Markov logic networks

Because Markov logic networks are a novel representation, there are many directions in which they may be further developed.

We plan to scale up inference in MLNs. Our current implementation uses the simplest form of Markov Chain Monte Carlo (MCMC) for inference (i.e., Gibbs sampling). Conveniently, MLNs may make use of any advances in Markov network inference. Of particular interest would be advanced MCMC techniques, such as the Swendsen-Wang algorithm [37]. For certain special cases, efficient graph-cut algorithms [55] may be used instead. By automatically identifying special cases and applying the best algorithm, MLNs can become a useful, general tool for probabilistic and logical inference. We envision MLNs being used similarly to databases: By supplying a standard,

powerful language for representing and manipulating data, databases have enabled the "front-end" to remain constant while improving the optimizations used in the "back-end". Similarly, MLNs' representation is powerful enough that the front-end may remain constant while the back-end is advanced by using optimization of special cases and recent developments in inference algorithms.

For the purposes of inference, we form the ground Markov network that that includes all nodes which may affect the query node. We plan to explore methods that only instantiate portions of this graph — those that are most likely to affect the answer. For example, we could instantiate a relevant subgraph, and use MCMC to sample the probability distribution entailed by this graph. Alternatively, each MCMC sample may be over a different subgraph. Such techniques will be especially useful in infinite domains.

Our use of inductive logic programming (ILP) methods with MLNs was very preliminary. Ideally, ILP methods would be fully incorporated into the learning process of MLNs, for the purposes of feature induction and knowledge base refinement. In particular, the ILP structure learning algorithms should have a function of the likelihood or pseudo-likelihood as the candidate scoring function.

8.2.3 Merging structure information

Recall that for merging structure information, we used a generative model for experts. This model was very simplistic. Clearly, experts do not add, remove, and reverse edges independently and with uniform probability. We plan to refine our model of experts to incorporate more complex behaviors. For example, one common mistake is to consider a node to be directly dependent on another when in fact it is only indirectly dependent through a third node. Further, some nodes are easier to understand and some are harder; edge errors are likely to be more common when the edges involve nodes that are harder to understand.

So far, we have only applied our approach to Bayesian networks. We would like to apply it to other representations (e.g. Markov networks) and also extend it to relational domains. In particular, it would be very useful to extend the techniques to merging "structural" information about Markov logic networks from multiple users.

8.2.4 Trust

In our work on trust, we assumed that statements are independent. We would like to investigate how dependencies between statements may be handled. For example, if we consider a taxonomy to be a set of class-subclass relationships, and consider each relationship to be an independent statement, then merging such taxonomy beliefs is not likely to lead to a useful taxonomy. We would like to be able to merge structural elements like taxonomies; Doan et al. [31, 29] may provide useful insights into possible solutions.

The path algebra and probabilistic interpretations were shown to be nearly identical, and the probabilistic interpretation is a generalization of PageRank. Considering PageRank works so well on Web pages, it would be interesting to apply the web of trust ideas ranking Web pages. For instance, we might find it useful to replace the sum with a maximum in PageRank. In general, we would like to consider networks in which not all users employ the same belief combination function, perhaps by modifying the global interpretation in order to relax the requirements put on the concatenation and aggregation functions.

There are many tradeoffs between computation, communication, and storage requirements for the different web of trust architectures (peer to peer, central server, hierarchical, etc.), algorithms (semi-naïve, Warshall, etc.), and strategies (merge beliefs on demand, store all beliefs, etc.). We would like to formalize these tradeoffs for better understanding of the efficiency of trust combination.

We considered only single-valued beliefs and trusts. In general, a belief could actually be multi-valued, representing a magnitude in multiple dimensions, such as “truth”, “importance”, and “novelty”. We would also like to consider multi-valued trusts, such as those used by Gil and Ratnakar [52], which may represent similar dimensions as beliefs (but applied to users). It may be possible to combine beliefs and trusts into one concept, ‘opinion’, which may be similarly applied to both statements and users. Similarly, we would also like to allow users to specify *topic-specific* trusts. With topic-specific trusts, the normalized sum combination function would probably be similar to query-dependent PageRank [113].

8.2.5 General

We have made no distinction between poor, low-quality contributors and malicious, intentionally bad ones. With enough training data, both may be discovered and handled, using machine learning techniques. However, it is likely that malicious contributors will act in a characteristic, detectable manner. Explicitly detecting and overcoming maliciousness could reduce the load on the machine learning algorithms, and the amount of data needed.

Though we have looked at input in the form of logical statements (either Horn clauses or Markov logic networks), and also structural information (dependencies between variables in a domain), there are still other forms of knowledge which we would like a collective knowledge base to accept. For example, simply clustering together predicates and variables that tend to be related could be useful, particularly in structural learning. We have looked at meta-information such as trust, but other forms of meta-information (e.g., domain size estimates, missing information, known experts, external references to knowledge in the collective knowledge base, etc.) could be equally useful. The ultimate goal is a collective knowledge base that accepts all forms of input, and uses them all in inference and learning.

Finally, although we have built a simple test system (BibServ), and have performed some experiments with volunteers, we have not yet created a large-scale collective knowledge base. In the near future, we plan to build a public Web site which will use all of the techniques described in this dissertation to enable contributors to build a large collective knowledge base. We look forward to the research challenges, seen and unforeseen, which await us.

END NOTES

1.1 Our description of Cyc is based on the available publications about it, made primarily in its founding years. Cyc has developed since, but these developments are not publicly available.

1.2 By one count, there are over 100,000 unique users who have replied to postings in newsgroups related to computers in the last year alone.

2.1 A range-restricted clause is one in which every variable in the head appears at least once in the body.

2.2 For the remainder of this thesis, we will always assume that conditional probability distributions are specified using CPTs.

2.3 This is true only if we assume all training data instances are independent of each other.

3.1 From http://commonsense.media.mit.edu.

3.2 By "flatten", we mean to convert relational knowledge into a propositional form. For example, if the knowledge base contains the fact $\texttt{Friends}(\texttt{Anna}, \texttt{Bob})$, this can be flattened to a single proposition $\texttt{Friends\_Anna\_Bob}$. Difficulties arise if the knowledge base contains facts such as $\forall \texttt{x}\ \texttt{Friends}(\texttt{Anna}, \texttt{x})$, in which case flattening results in many propositions, one for each person in the knowledge base.

3.3 Note that, although this learning process involves constructing a network for each training example, these networks only need to be constructed once. When there are missing values in the training set, the time spent constructing the BNs is insignificant compared to the time spent iterating with EM [77].

4.1 Rules for a topic are defined as rules whose consequent is a predicate belonging to that topic.

4.2 Experiments were run on a 1Ghz Pentium 3 computer with 512Mb of RAM.

4.3 Available at http://www.cs.huji.ac.il/labs/compbio/Repository/networks.html.

4.4 This can be higher than the average accuracy of the experts, if different experts answer different questions, because an unanswered query is counted as answered incorrectly.

5.1 In practice, we have found it useful to augment the MLN with a unit clause for each predicate. Roughly speaking, the weight of a unit clause can capture the marginal distribution of the corresponding predicate, leaving the weights of the non-unit clauses free to model only dependencies between predicates.

5.2 This translation is the same as would be used to convert a Bayesian network to a Markov network. A Bayesian network is defined by a product of conditional probabilities, and a Markov network is defined by an exponential of a sum of weights. The latter can be rewritten as a product of exponentiated weights. Thus, if each weight is the logarithm of a conditional probability, the Markov network will be equivalent to a product over conditional probabilities.

5.3 Use of SQL aggregates requires that their definitions be imported into the MLN.

5.4 If a query predicate contains variables, it is replaced by all its possible groundings.

5.5 http://www.cs.washington.edu/homes/kautz/walksat/

5.6 Inspection reveals that the smaller drop-offs in precision at very low recalls are due to students who graduated or changed advisors after co-authoring many publications with them.

6.1 The *a posteriori* estimates, after we have seen E, can be different for different (i, j). However, making the usual parameter independence assumption [61], observation of one e_{ij} does not affect estimation of any other. (Notice that, in our treatment, E is composed of exactly one sample of each e_{ij} variable.)

6.2 Sequential search performed slightly better than Powell's method, and is the one we report. Both methods take negligible time compared to the structure learning.

6.3 This is equivalent to the negative log-likelihood on these samples, minus the entropy of the true distribution estimated from the samples.

6.4 Notice that the cross-validation for κ was performed with structure search, while for the parameters of our expert model it was only performed with parameter learning, the results being used for structure learning as well.

6.5 Notice K-L distances for "true" are not zero, because they refer to *learning* starting from the true structure.

7.1 Trust is, of course, a complex and multidimensional phenomenon, but we make a start by embodying it in a single numeric coefficient per user-source pair.

7.2 While this may not guarantee the probabilistic soundness of the resulting beliefs, we believe it is necessary for scalability on the size of the Web, and our experiments indicate it still produces useful results. Scalable probabilistic approximations are a direction for future research.

7.3 The trust relationships can be obtained by crawling the site, as described in Richardson and Domingos [114]. Although the full graph contains 75,000 users, we restricted our experiments to the first 5000 users (by crawl-order), which formed a network of 180,000 edges.

7.4 Incorporating traditional measures of query relevance (for instance, TFIDF) may lead to a better ordering of entries. One probabilistic technique for this is that of query-dependent PageRank [113].

BIBLIOGRAPHY

[1] A. Abdul-Rahman and S. Hailes, "A distributed trust model," in *Proceedings of New Security Paradigms Workshop*, 1997, pp. 48–60.

[2] R. Agrawal, S. Dar, and H. V. Jagadish, "Direct transitive closure algorithms: Design and performance evaluation." *ACM Transactions on Database Systems*, vol. 15, no. 3, pp. 427–458, 1990.

[3] R. Agrawal and H. V. Jagadish, "Multiprocessor transitive closure algorithms." in *Proceedings of the International Symposium on Databases in Parallel and Distributed Systems*, Austin, TX, 1988, pp. 56–66.

[4] A. Agresti, *Categorical Data Analysis*. New York, NY: Wiley, 1990.

[5] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[6] C. Anderson, P. Domingos, and D. Weld, "Relational Markov models and their application to adaptive Web navigation," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Edmonton, Canada: ACM Press, 2002, pp. 143–152.

[7] F. Bacchus, *Representing and Reasoning with Probabilistic Knowledge*. Cambridge, MA: MIT Press, 1990.

[8] F. Bancilhon, "Naive evaluation of recursively defined relations." in *On Knowledge Base Management Systems (Islamorada)*, 1985, pp. 165–178.

[9] R. Bellman and M. Giertz, "On the analytic formalism of the theory of fuzzy sets." *Information Sciences*, vol. 5, pp. 149–156, 1973.

[10] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, pp. 34–43, 2001.

[11] J. Besag, "Statistical analysis of non-lattice data," *The Statistician*, vol. 24, pp. 179–195, 1975.

[12] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, Oakland, CA, 1996, pp. 164–173.

[13] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.

[14] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," in *Proceedings of the Seventh International World Wide Web Conference*. Brisbane, Australia: Elsevier, 1998.

[15] B. Carre, *Graphs and Networks*. Oxford: Claredon Press, 1978.

[16] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan, "Automatic resource compilation by analyzing hyperlink structure and associated text," in *Proceedings of the Seventh International World Wide Web Conference*. Brisbane, Australia: Elsevier, 1998, pp. 65–74.

[17] D. M. Chickering and D. Heckerman, "Efficient approximations for the marginal likelihood of Bayesian networks with hidden variables," *Machine Learning*, vol. 29, pp. 181–212, 1997.

[18] C. K. Chow and C. N. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE Transactions on Information Theory*, vol. 14, pp. 462–467, 1968.

[19] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of COmputing*, 1971, pp. 151–158. [Online]. Available: http://theory.lcs.mit.edu/ dmjones/STOC/stoc71.html

[20] G. F. Cooper, “The computational complexity of probabilistic inference using Bayesian belief networks.” *Artificial Intelligence*, vol. 42, no. 2-3, pp. 393–405, 1990.

[21] V. S. Costa, D. Page, M. Qazi, , and J. Cussens, “CLP(BN): Constraint logic programming for probabilistic knowledge,” in *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*. Acapulco, Mexico: Morgan Kaufmann, 2003, pp. 517–524.

[22] C. Cumby and D. Roth, “Feature extraction languages for propositionalized relational learning,” in *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*. Acapulco, Mexico: IJCAII, 2003, pp. 24–31.

[23] J. Cussens, “Loglinear models for first-order probabilistic reasoning,” in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*. Stockholm, Sweden: Morgan Kaufmann, 1999, pp. 126–133.

[24] J. Cussens, “Individuals, relations and structures in probabilistic models,” in *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*. Acapulco, Mexico: IJCAII, 2003, pp. 32–36.

[25] J. Cussens, “Parameter estimation in stochastic logic programs.” *Machine Learning*, vol. 44, no. 3, pp. 245–271, 2001.

[26] L. De Raedt and L. Dehaspe, “Clausal discovery,” *Machine Learning*, vol. 26, pp. 99–146, 1997.

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

[28] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” *Journal of the Royal Statistical Society, Series B*, vol. 39, pp. 1–38, 1977.

[29] A. Doan, P. Domingos, and A. Halevy, "Reconciling schemas of disparate data sources: A machine-learning approach," in *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*. Santa Barbara, CA: ACM Press, 2001, pp. 509–520.

[30] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy, "Learning to match ontologies on the Semantic Web," *VLDB Journal*, 2004, to appear.

[31] A. Doan, J. Madhavan, P. Domingos, and A. Halevy, "Learning to map between ontologies on the Semantic Web," in *Proceedings of the Eleventh International World Wide Web Conference*. Honolulu, HI: ACM Press, 2002, pp. 662–673.

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

[33] P. Domingos and M. Richardson, "Mining the network value of customers," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco, CA: ACM Press, 2001, pp. 57–66.

[34] D. Dou, D. McDermott, and P. Qi, "Ontology translation on the Semantic Web," in *Proceedings of the International Conference on Ontologies, Databases and Applications of Semantics*, 2003.

[35] S. Dzeroski, L. de Raedt, and S. Wrobel, Eds., *Proceedings of the KDD-2002 Workshop on Multi-Relational Data Mining*. Edmonton, Canada: ACM Press, 2002.

[36] S. Dzeroski, L. de Raedt, and S. Wrobel, Eds., *Proceedings of the Second International Workshop on Multi-Relational Data Mining*. Washington, DC: ACM Press, 2003.

[37] R. Edwards and A. Sokal, "Generalization of the Fortuin-Kasteleyn-Swendsen-Wang representation and Monte Carlo algorithm," in *Physics Review D*, vol. 38, 1988, pp. 2009–2012.

[38] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates, "Web-scale information extraction in KnowItAll," in *Thirteenth International World Wide Web Conference*, 2004, pp. 100–110.

[39] M. Frauenfelder, "Revenge of the know-it-alls: Inside the Web's free-advice revolution," *Wired*, vol. 8, no. 7, pp. 144–158, 2000.

[40] S. French, "Group consensus probability distributions: A critical survey," in *Bayesian Statistics 2*, J. M. Bernardo, M. H. DeGroot, D. V. Lindley, and A. F. M. Smith, Eds. Amsterdam, Netherlands: Elsevier, 1985, pp. 183–202.

[41] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proceedings of the Thirteenth International Conference on Machine Learning*. Bari, Italy: Morgan Kaufmann, 1996, pp. 148–156.

[42] N. Friedman, "The Bayesian structural EM algorithm," in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*. Madison, WI: Morgan Kaufmann, 1998, pp. 129–138.

[43] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer, "Learning probabilistic relational models," in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. Stockholm, Sweden: Morgan Kaufmann, 1999, pp. 1300–1307.

[44] A. Gelman and X.-L. Meng, "Simulating normalizing constants: From importance sampling to bridge sampling to path sampling," *Statistical Science*, vol. 13, no. 2, pp. 163–185, 1998.

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

[46] C. Genest and J. V. Zidek, "Combining probability distributions: A critique and an annotated bibliography," *Statistical Science*, vol. 1, pp. 114–148, 1986.

[47] L. Getoor and D. Jensen, Eds., *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*. Austin, TX: AAAI Press, 2000.

[48] L. Getoor and D. Jensen, Eds., *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*. Acapulco, Mexico: IJCAII, 2003.

[49] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer, "Learning Probabilistic Relational Models," in *Relational Data Mining*, S. Dzeroski and N. Lavrac, Eds. Springer-Verlag, 2001, pp. 307–333.

[50] L. Getoor, N. Friedman, D. Koller, and B. Taskar, "Learning probabilistic models of link structure," *Journal of Machine Learning Research*, vol. 3, pp. 679–707, 2002.

[51] C. J. Geyer and E. A. Thompson, "Constrained monte carlo maximum likelihood for dependent data," *Journal of the Royal Statistical Society, Series B*, vol. 54, no. 3, pp. 657–699, 1992.

[52] Y. Gil and V. Ratnakar, "Trusting information sources one citizen at a time." in *International Semantic Web Conference*, Sardinia, Italy, 2002, pp. 162–176.

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

[54] W. R. Gilks, A. Thomas, and D. J. Spiegelhalter, "A language and program for complex Bayesian modelling," *The Statistician*, vol. 43, pp. 169–78, 1994.

[55] D. Greig, B. Porteous, and A. Seheult, "Exact maximum a posteriori estimation for binary images," *Journal of the Royal Statistical Society, Series B*, vol. 51, no. 2, pp. 271–279, 1989.

[56] R. Guha, "Open rating systems," Stanford Knowledge Systems Laboratory, Stanford, CA, Tech. Rep., 2003.

[57] J. Halpern, "An analysis of first-order logics of probability," *Artificial Intelligence*, vol. 46, pp. 311–350, 1990.

[58] M. S. Handcock, "Assessing degeneracy in statistical models of social networks," Center for Statistics and the Social Sciences, University of Washington, Seattle, WA, Working Paper 39, 2003.

[59] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 993–1000, 1990.

[60] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie, "Dependency networks for inference, collaborative filtering, and data visualization," *Journal of Machine Learning Research*, vol. 1, pp. 49–75, 2000.

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

[62] D. Heckerman, "A tutorial on learning Bayesian networks," Microsoft Research, Redmond, WA, Tech. Rep. MSR-TR-95-06, Mar. 1995.

[63] R. A. Howard, "Information value theory," *IEEE Transactions on Systems Science and Cybernetics*, vol. 2, pp. 22–26, 1966.

[64] G. Hulten, D. M. Chickering, and D. Heckerman, "Learning Bayesian networks from dependency networks: A preliminary study," in *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, Key West, FL, 2003.

[65] M. Jaeger, "Reasoning about infinite random structures with relational Bayesian networks," in *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*. Trento, Italy: Morgan Kaufmann, 1998.

[66] M. Jaeger, “Constraints as data: A new perspective on inferring probabilities,” in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*. Seattle, WA: Morgan Kaufmann, 2001, pp. 755–760.

[67] T. Joachims, “A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization,” in *Proceedings of the Fourteenth International Conference on Machine Learning (ICML-97)*. San Francisco, CA: Morgan Kaufmann, 1997, pp. 143–151.

[68] S. Kamvar, M. Schlosser, and H. Garcia-Molina, “The eigentrust algorithm for reputation management in p2p networks,” in *Proceedings of the Twelfth International World Wide Web Conference*, 2003.

[69] D. Karger and N. Srebro, “Learning Markov networks: maximum bounded tree-width graphs,” in *Symposium on Discrete Algorithms*, 2001, pp. 392–401. [Online]. Available: citeseer.nj.nec.com/karger01learning.html

[70] H. Kautz and B. Selman, “Planning as satisfiability,” in *Proceedings of the Tenth European Conference on Artificial Intelligence*. Vienna, Austria: Chichester, UK: John Wiley & Sons, 1992, pp. 359–363.

[71] H. Kautz, B. Selman, and M. Shah, “ReferralWeb: Combining social networks and collaborative filtering,” *Communications of the ACM*, vol. 40, no. 3, pp. 63–66, 1997.

[72] K. Kersting, “Bayesian logic programs,” Ph.D. dissertation, University of Freiburg, Freiburg, Germany, 2000.

[73] K. Kersting and L. De Raedt, “Towards combining inductive logic programming with Bayesian networks,” in *Proceedings of the Eleventh International Conference on Inductive Logic Programming*. Strasbourg, France: Springer, 2001, pp. 118–131.

[74] K. Kersting, L. D. Raedt, and S. Kramer, “Interpreting Bayesian Logic Programs,” in *Working Notes of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, 2000.

[75] J. M. Kleinberg, “Authoritative sources in a hyperlinked environment,” in *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. Baltimore, MD: ACM Press, 1998, pp. 668–677.

[76] D. Koller, A. Levy, and A. Pfeffer, “P-Classic: A tractable probabilistic description logic,” in *Proceedings of the Fourteenth National Conference on Artificial Intelligence*. Providence, RI: AAAI Press, 1997, pp. 390–397.

[77] D. Koller and A. Pfeffer, “Learning probabilities for noisy first-order rules,” in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*. Nagoya, Japan: Morgan Kaufmann, 1997, pp. 1316–1321.

[78] D. Koller and A. Pfeffer, “Probabilistic frame-based systems,” in *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. Madison, WI: AAAI Press, 1998, pp. 580–587.

[79] C. C. T. Kwok, O. Etzioni, and D. S. Weld, “Scaling question answering to the web,” in *Proceedings of the Tenth International World Wide Web Conference*, 2001, pp. 150–161.

[80] K. Lari and S. J. Young, “The estimation of stochastic context-free grammars using the Inside-Outside algorithm,” *Computer Speech and Language*, vol. 4, pp. 35–56, 1990.

[81] N. Lavrac and S. Dzeroski, *Inductive Logic Programming: Techniques and Applications*. Chichester, UK: Ellis Horwood, 1994.

[82] D. B. Lenat and R. V. Guha, *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Reading, MA: Addison-Wesley, 1990.

[83] D. V. Lindley, “Reconciliation of discrete probability distributions,” in *Bayesian Statistics 2*, J. M. Bernardo, M. H. DeGroot, D. V. Lindley, and A. F. M. Smith, Eds. Amsterdam, Netherlands: Elsevier, 1985, pp. 375–390.

[84] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 3, pp. 503–528, 1989.

[85] J. W. Lloyd, *Foundations of Logic Programming*. Berlin, Germany: Springer, 1987.

[86] E. Lloyd-Richardson, A. Kazura, C. Stanton, R. Niaura, and G. Papandonatos, "Differentiating stages of smoking intensity among adolescents: Stage-specific psychological and social influences," *Journal of Consulting and Clinical Psychology*, vol. 70, no. 4, 2002.

[87] 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*. Madison, WI: Morgan Kaufmann, 1998, pp. 359–367.

[88] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder, "An environment for merging and testing large ontologies," in *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning*. Breckenridge, CO: Morgan Kaufmann, 2000.

[89] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge University Press, 1995.

[90] S. Muggleton, "Stochastic logic programs," in *Advances in Inductive Logic Programming*, L. de Raedt, Ed. Amsterdam, Netherlands: IOS Press, 1996, pp. 254–264.

[91] S. Muggleton, "Learning stochastic logic programs," in *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, 2000. [Online]. Available: citeseer.nj.nec.com/muggleton00learning.html

[92] S. Muggleton, "Semantics and derivation for stochastic logic programs," in *Proceedings of the UAI-2000 workshop on Knowledge-Data Fusion*, 2000.

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

[94] S. Muggleton, “Inverse entailment and Progol,” *New Generation Computing, Special issue on Inductive Logic Programming*, vol. 13, no. 3-4, pp. 245–286, 1995.

[95] K. P. Murphy, Y. Weiss, and M. I. Jordan, “Loopy belief propagation for approximate inference: An empirical study,” in *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence*, 1999, pp. 467–475.

[96] J. Neville and D. Jensen, “Collective classification with relational dependency networks,” in *Proceedings of the Second International Workshop on Multi-Relational Data Mining*, S. Dzeroski, L. de Raedt, and S. Wrobel, Eds. Washington, DC: ACM Press, 2003, pp. 77–91.

[97] R. T. Ng and V. S. Subrahmanian, “Probabilistic logic programming,” *Information and Computation*, vol. 101, no. 2, pp. 150–201, December 1992. [Online]. Available: http://theory.lcs.mit.edu/ iandc/ic92.html

[98] L. Ngo and P. Haddawy, “Answering queries from context-sensitive probabilistic knowledge bases,” *Theoretical Computer Science*, vol. 171, pp. 147–177, 1997.

[99] L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank citation ranking: Bringing order to the web,” Stanford University, Stanford, CA, Tech. Rep., 1998.

[100] Parag and P. Domingos, “Collective object identification,” Department of Computer Science and Engineering, University of Washington, Seattle, WA, Tech. Rep., 2004, http://www.cs.-washington.edu/homes/pedrod/coi.pdf.

[101] M. Pazzani and D. Kibler, “The utility of knowledge in inductive learning,” *Machine Learning*, vol. 9, pp. 57–94, 1992.

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

[103] D. Pennock and M. Wellman, "Graphical representations of consensus belief," in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*. Stockholm, Sweden: Morgan Kaufmann, 1999, pp. 531–540.

[104] D. M. Pennock, F. A. Nielsen, and C. L. Giles, "Extracting collective probabilistic forecasts from Web games," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco, CA: ACM Press, 2001, pp. 174–183.

[105] M. P. Perrone and L. M. Cooper, "When networks disagree: Ensemble methods for hybrid neural networks," in *Artificial Neural Networks for Speech and Vision*, R. J. Mammone, Ed. London, UK: Chapman and Hall, 1993, pp. 126–142.

[106] A. Popescul and L. H. Ungar, "Structural logistic regression for link analysis," in *Proceedings of the Second International Workshop on Multi-Relational Data Mining*, S. Dzeroski, L. de Raedt, and S. Wrobel, Eds. Washington, DC: ACM Press, 2003, pp. 92–106.

[107] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, 2nd ed. Cambridge, UK: Cambridge University Press, 1992.

[108] A. Puech and S. Muggleton, "A comparison of stochastic logic programs and Bayesian logic programs," in *Proceedings of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*. Acapulco, Mexico: IJCAII, 2003, pp. 121–129.

[109] J. R. Quinlan, "Learning logical definitions from relations," *Machine Learning*, vol. 5, pp. 239–266, 1990.

[110] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.

[111] E. S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol, CA: O'Reilly, 1999.

[112] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: An open architecture for collaborative filtering of netnews," in *Proceedings of the ACM 1994 Conference on Computer Supported Cooperative Work*. New York, NY: ACM Press, 1994, pp. 175–186.

[113] M. Richardson and P. Domingos, "The intelligent surfer: Probabilistic combination of link and content information in PageRank," in *Advances in Neural Information Processing Systems 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. Cambridge, MA: MIT Press, 2002, pp. 1441–1448.

[114] M. Richardson and P. Domingos, "Mining knowledge-sharing sites for viral marketing," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Edmonton, Canada: ACM Press, 2002, pp. 61–70.

[115] S. Riezler, "Probabilistic constraint logic programming," Ph.D. dissertation, Universitat Tubingen, 1998.

[116] D. Roth, "On the hardness of approximate reasoning," *Artificial Intelligence*, vol. 82, pp. 273–302, 1996.

[117] S. Sanghai, P. Domingos, and D. Weld, "Dynamic probabilistic relational models," in *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. Acapulco, Mexico: Morgan Kaufmann, 2003, pp. 992–997.

[118] T. Sato and Y. Kameya, "PRISM: A symbolic-statistical modeling language," in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*. Nagoya, Japan: Morgan Kaufmann, 1997, pp. 1330–1335.

[119] B. Selman, H. Kautz, and B. Cohen, "Local search strategies for satisfiability testing," in *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, D. S. Johnson and M. A. Trick, Eds. Washington, DC: American Mathematical Society, 1996, pp. 521–532.

[120] P. Singh, “The public acquisition of commonsense knowledge,” in *Proceedings of the AAAI Spring Symposium on Acquiring (and Using) Linguistic (and World) Knowledge for Information Access*. Palo Alto, CA: AAAI Press, 2002.

[121] D. G. Stork, “Using open data collection for intelligent software,” *IEEE Computer*, vol. 33, no. 10, pp. 104–106, 2000.

[122] B. Taskar, P. Abbeel, and D. Koller, “Discriminative probabilistic models for relational data,” in *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*. Edmonton, Canada: Morgan Kaufmann, 2002, pp. 485–492.

[123] A. Tversky and D. Kahneman, “Judgment under uncertainty: Heuristics and biases,” *Science*, vol. 185, pp. 1124–1131, 1974.

[124] S. Warshall, “A theorem on boolean matrices.” *Journal of the ACM*, vol. 9, no. 1, pp. 11–12, 1962.

[125] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications*. Cambridge, UK: Cambridge University Press, 1994.

[126] M. Wellman, J. S. Breese, and R. P. Goldman, “From knowledge bases to decision models,” *Knowledge Engineering Review*, vol. 7, 1992.

[127] D. Wolpert, “Stacked generalization,” *Neural Networks*, vol. 5, pp. 241–259, 1992.

Appendix A

ADDENDUM TO CHAPTER 4 EXPERIMENTAL SECTION

This appendix contains the instructions which were given to the four computer “experts”, for the experiment in Section 4.4.2.

A.1 Email Sent to Volunteers

Here’s what I would like you to do. I don’t expect (or want) you to use all of the variables in the domain. I’m going to give you 30 minutes to write rules. That means, I’d prefer you write fewer, good rules than cram as many cruddy rules down as you can. Can you please mark your progress each 15 minutes?

Here’s my suggestion: use some time to glance through the variables to see what all they are about. Then, choose some sort of area you’d like to concentrate on and go through again marking all the variables you think are relevant, then put them into rules. If you have time left, you can do a new area, or make some rules for general areas, or whatever. This is just a suggestion – feel free to do whatever you want.

Finally, I don’t know how many you can finish in 30 minutes. If 30 seems way too short, please feel free to go on to like 60 minutes, as long as you keep marking each 15 minutes where you are. I’m looking for about a third to half the variables being used.

Thanks again, I really do appreciate it! Matt

A.2 Directions Given to Volunteers

This document describes the Windows 95 printer domain task. Please feel free to consult any source of knowledge (e.g. the internet, your printer manual, etc…) in performing the task. In the table below you'll find a list of variables (such as `DskLocal`), and a description of what each variable means ("Is there enough local disk space"). Also supplied are the meanings for the variable in the case of it being true or false. In almost all cases, the "True" meaning is the good meaning (e.g. There *is* enough local disk space). There are a number of mostly observation type variables, a number of intermediate variables, and a few final result variables. Your job is to write a knowledge base (a set of rules) that describes some of the printer domain. Rules must all be of the form :

```
p ant1 ^ ant2 ^ … ^ antN => cons
```

Where ant1…antN are the N antecedents, and cons is the consequent, and the meaning is "I am *p* confident that if ant1 and ant2 and … and antN, then cons". An antecedent may be any of the variables, possibly negated. The consequent may be any of the variables, but must be in the positive sense. A rule may have any number of antecedents, from 0 to the number of variables in the domain. Each rule is also prefixed by *p*, a number from 0 to 1, which is your confidence that you got the rule right. Here is an example rule:

```
0.8 PrtPaper ^ !PrtSpool => LclOK
```

Which means (see table below) "If there is paper in the paper tray, and you are not using print spooling, then the local print connection is working, and I am 80% confident that this rule is accurate". (Obviously a bogus rule, but I wouldn't want to bias you with some real rules, now would I?!).

Assume that each variable is in its false state, unless some rule makes it true. For example, if you don't have any rules that imply "PrtPaper" then you are implicitly stating that the printer is always out of paper. Note, sometimes some things have some probability of being true without a cause (a prior probability of being true). For instance, we know smoking causes cancer, but how often is smoking itself true? For this, remember that it is okay to write rules without antecedents, such as "0.3 => smoking". For these rules, the number *p* that you provide is your estimate of the probability that consequent is true (as opposed to your confidence that smoking is always true)

In some cases, it may be hard to tell if you should write (A => B) or (B => A). Please try to write the rules in a causal direction -- e.g. if you write A => B it should generally mean that A causes B. For instance, "smoking => cancer" is correct while "cancer => smoking" is not.

Finally, please try to use intermediate variables to simplify your rules when you can. I recommend writing down the variables on a piece of paper in roughly the order you think they are from cause to effect, then crossing them out or drawing lines between them as you write rules, to help ensure you use most, if not all, of them, and that your logic statements can link together to form longer chains of reasoning.

There are two ways to think of things -- 1) The printer works and write rules explaining when things will fail, or 2) The printer doesn't work and write rules explaining when it will. The first is maybe more natural since we are used to printers generally working, but for this task, you have to think of things in the second way. Since everything is false, by default it means the printer doesn't work, and you have to explain when it will work.

Finally, note that if you want to say "if A or B then C", you can do this by just writing two rules, "A => C" and "B => C" – rules are automatically or'd together.

Thank you so much for helping me out. For any questions, please email me!
Matt Richardson (mattr@cs)

Note, please be careful with `NetPrint`, which, despite the name, means you are printing locally when it is true, and printing over the network when it is false. Please be careful to use it correctly!. There are a number of variables who's names are confusing, like `GrbldPS`, which is true when the output is *not* garbled, and the problem variables, which are true when there is no problem. **It is always the case that the True (T) sense of the variable is the "good" meaning for it and the False (F) sense is the "bad" meaning**. (except in one or two cases like whether you are using DOS or Windows).

Variable	Short Description	T	F	Longer Description
Easily Observable				
DskLocal	Local Disk Space	Greater than 2 Mb	Less than 2 Mb	Is there more than 2 Megabytes free on the local hard drive?
PrtSpool	Print Spooling	Enabled	Disabled	Is Printer Spooling enabled?
PrtCbl	Local Printer Cable	Connected	Loose	Is the local printer cable (the one running from the parallel port in the back of the computer to the printer) firmly connected?
PrtOn	Printer On and Online	Yes	No	Is the printer on and ready (online)?
PrtDriver	Correct Driver	Yes	No	Is the correct printer driver installed?
PrtPaper	Printer Paper Supply	Has Paper	No Paper	Does the printer have paper?
DrvSet	Driver Configuration	Correct	Incorrect	Is the driver properly configured?
NetPrint	Printing Locally	Yes (Local printer)	No (Network printer)	Is the printer connected by a local cable directly to the computer or over the network?
PrtPath	Net Printer Pathname	Correct	Incorrect	Networked printer path is correct
PrtSel	Correct Printer Selected	Yes	No	Have you chosen to print to the correct printer?
PrtMem	Printer Memory	Greater than 2 Mb	Less than 2Mb	Does the printer have more then 2 Megabytes of memory?
PrtPort	Correct Local Port	Yes	No	For a local printer, is the correct port (parallel port) chosen?
DSApplctn	Print Environment	DOS	Windows	Are you trying to print from DOS or Windows?
PrtMpTPth	Port Mapping to Path	Correct	Incorrect	Is the mapping from port to printer path correct?
PgOrnttnOK	Page Orientation	Correct	Incorrect	Did it print in the correct orientation (e.g. landscape vs. portrait)?
PrntngArOK	Printer Printing Area	Correct	Incorrect	Does the document fit in the area the printer can print?
ScrnFntNtPrntrFnt	Screen Matches Printer	Yes	No	Do the fonts onscreen match the fonts that were printed?
GrphcsRltdDrvrSttngs	Driver Config- Graphics	Correct	Incorrect	Are the graphics settings in the driver configuration correct?
TrTypFnts	True Type Fonts	Yes	No	Are you using True Type Fonts? (special fonts that can be scaled to any size)
FntInstlltn	Font Installation	Verified	Faulty	Are fonts properly installed?
PrntrAccptsTrtyp	Printer Accepts Truetype	Yes	No	Does the printer accept True Type fonts?
AppDtGnTm	App Data Generation	Fast Enough	Too Long	Does the application take the right amount of time to send the data to the printer

HrglssDrtnAftrPrnt	Hourglass Duration	Fast Enough	Too Long	How long does the hourglass appear when you choose to print?
REPEAT	Repeatable Problem	Yes	No	Is the problem repeatable (vs. sporatic)?
PrtPScript	Postscript Printer	Yes	No	Are you using a Postscript printer?
PSERRMEM	PS Error Memory	No Error	Low Memory	Is it fine, or do you get a low memory error?
TstpsTxt	testps.txt Output	Yes	No	Printing a small test.txt file to a postscript printer works
EPSGrphc	EPS Graphic	Normal (TIF, BMP),	EPS	Does your document contain normal images, or EPS (Encapsulated Postscript) images?
AppOK	Application	Correct	Incorrect / Corrupt	Is the application you are printing from working properly?
DataFile	Document	Correct	Incorrect / Corrupt	Is the datafile you loaded into the application okay?
PrtFile	Print to File	Yes	No	Can you successfully print to a file?
PrtIcon	Printer Icon	Normal	Grayed Out	Does the windows printer icon look normal, or is it disabled (grayed out)
PrtQueue	Printer Queue	Short	Long	The printer queue is not too long
TnrSpply	Toner Supply	Adequate	Low	Is there enough toner in the toner tray?
PTROFFLINE	Printer Driver Set Online	Online	Offline	Is the printer driver set online?

More Intermediate type variables				
PrtStatPaper	Printer Status	No Error	Jam, Out, Bin Full	Does the printer say no error, or does it have a paper jam / the paper bin is full or out of paper
PrtStatToner	Printer Status	No Error	Low, None	Does the printer say no error, or that it is low on toner?
PrtStatMem	Printer Status	No Error	Out of Memory	Does the printer say no error, or that it is out of memory
PrtStatOff	Printer Status	No Error	OFFLINE, OFF	Is the printer on/online or is the printer off / offline (the ONLINE LED is off)?
PrtThread	Port thread/Prt Proc OK	OK	Corrupt / Buggy	The printer spooling process is ok
AppData	Application Data	Correct	Incorrect or corrupt	The application is outputting proper print data
PrtDataOut	PrintDataOut	Yes	No	The data to be sent to the printer is ok
PC2PRT	PC to PRT Transport	Yes	No	The path from computer to printer is okay -- e.g. the printer gets the data
NetOK	NET OK	Yes	No	Is the network working okay (in windows)?
GDIOUT	GDI Output OK	Yes	No	The driver output is okay
EMFOK	EMF OK	Yes	No	The Enhanced Metafile data sent to the driver is okay
GDIIN	GDI Input OK	Yes	No	The input to the driver is okay
DrvOK	Driver File Status	Reinstalled	Corrupt	The driver is installed okay
LclOK	LOCAL OK	Yes	No	Is the local print connection working? (in windows)
CmpltPgPrntd	Non PS Complete	Yes	No	The complete page is printed (if using a non-postscript printer)
TTOK	TT OK	Yes	No	TrueType fonts output correctly
GrbldOtpt	Non PS Not Garbled	Fine	Garbled	The output is fine (not garbled) (on a non-postscript printer)
NtwrkCnfg	Network Configuration	Correct	Incorrect	The network configuration is correct
NnTTOK	Non TT OK	Yes	No	Non TrueType fonts output correctly

DeskPrntSpd	Desk Speed	OK	Too Slow	The desktop computer is fast enough
CblPrtHrdwrOK	Cable/Port Hardware	Operational	Not Operational	The cable and local printer port hardware are okay
DS_NTOK	DOS-NET OK	Yes	No	In DOS, when printing over the network, the data is sent to the printer okay
DS_LCLOK	DOS-LOCAL OK	Yes	No	In DOS, when printing locally, the data is sent to the printer okay
PrntPrcssTm	Print Processing	Fast Enough	Too Long	Does the printer take the right amount of time to print?
LclGrbld	Local Garbled OK	Yes	No	Output is okay (not garbled) when printing locally
NtGrbld	Net Garbled OK	Yes	No	Output is okay (not garbled) when printing over a network
NnPSGrphc	Non PS Graphic	Yes	No	graphics print properly on a non-postscript printer
PrtTimeOut	Printer Timeouts	Long Enough	Too Short	How long does it take for the application to say that it can't print
AvlblVrtlMmry	Printer Virtual Mem	Adequate (> 1Mb)	Inadequate (< 1 Mb)	How much available virtual memory does the printer have?
GrbldPS	PS Not Garbled	Fine	Garbled	Output is okay (not garbled) when printing on a postscript printer
IncmpltPS	PS Complete	Yes	No	When you try to print to postscript printer, does it print the entire document?
PSGRAPHIC	PS Graphic	Yes	No	graphics print properly on a postscript printer
FllCrrptdBffr	Print Buffer	Intact (not Corrupt)	Full or Corrupt	The print buffer is okay
NtSpd	Net Speed	OK	Slow	The network speed is okay

Problems

Problem1	Have Output	Yes	No Output	Did it output okay?
Problem2	Speed Normal	Yes	Too Long	When you print, does it take a normal amount of time, or much longer than it should?
Problem3	Complete pages	Yes	Incomplete	Does it print full pages (vs. incomplete pages)
Problem4	Graphics fine	Yes	No	Are the graphics fine, or are they distorted or incomplete?
Problem5	Fonts fine	Yes	No	Did fonts appear fine, or are some or all of the fonts missing or appear incorrect or distorted
Problem6	Output Looks normal	Yes	Garbled	Is your output okay, or is it garbled?

Appendix B

ADDENDUM TO CHAPTER 5 EXPERIMENTAL SECTION

In this appendix, I provide additional information regarding the experiments described in Section 5.6:

- Directions Given to the Volunteers (Section B.1)
- Knowledge Base Produced by Volunteers (Section B.2)
- Algorithm Parameter Settings Used in the Experiments (Section B.3)

These details may also be found online at http://www.cs.washington.edu/ai/mln.

B.1 Directions Given to the Volunteers.

Thanks for helping me out on my research. Let's get straight to the task: I am asking you to write first-order statements which describe our department. First, let me describe the domain. The domain contains the following predicates:

Professor	(Person)	
Position	(Person, Position)	
Student	(Person)	
Phase	(Person, Phase)	
YearsInProgram	(Person, Integer)	
AdvisedBy	(Person X, Person Y)	"X is advised by Y"
TempAdvisedBy	(Person X, Person Y)	"X's temporary advisor is Y"
Publication	(Paper P, Person X)	"Person X is an author of paper P"
TaughtBy	(Course, Person, Quarter)	
TA	(Course, Person, Quarter)	
CourseLevel	(Course, Level)	
SamePerson	(Person, Person)	
SamePaper	(Paper, Paper)	
SameCourse	(Course, Course)	
SameProject	(Project, Project)	
SameQuarter	(Quarter, Quarter)	
SamePosition	(Position, Position)	
SamePhase	(Phase, Phase)	
SameInteger	(Integer, Integer)	

In general, for most of these predicates, you should not need to refer to specific ground constants. That is, I don't want statements referring to Publication(T,mattr) or things like that. There are some exceptions. Here are some constants you may or may not find useful:

Phase: Refers to the student's phase in grad school. There are three phases:
- post_Quals
- pre_Quals
- post_Generals

Level: Refers to the level of the course. There are four levels:

level_100	(intro courses 100, 142, 143)
level_300	(undergrad courses – junior level)
level_400	(advanced undergrad courses – senior level)
level_500	(grad level courses)

Position: Refers to the type of faculty for various professors

faculty
faculty_affiliate
faculty_adjunct
faculty_emeritus
faculty_visiting

Integer: Just used to say how many years a person has been in the program. They are year_1, year_2, …, year_12. (no jokes about YearsInProgram(mattr…)!)

The remaining types are pretty obvious. Here's a short description of each:
Person: includes professors and students
Paper: a conference or journal publication.
Course: Classes offered by the department
Quarter: identifier for the year and quarter (like autumn_0203).

Note, when you are doing the task, if you think there is some other predicate that would be useful for you, let me know. I may be able to add it to the domain pretty easily.

The Task

Please write down a first-order theory describing the domain. For simplicity, write it as a bunch of first-order statements, which are implicitly conjoined together. A statement may be anything you want which uses the above predicates. If not specified, variables are assumed to be universally quantified.

What I am looking for is a set of statements that roughly characterize our (or any) computer science department. The rules you write do not need to always be true. As long as they are generally more true than not, they will be useful.

Let me give you an example based on a "friendship" domain. You can write a simple statement like:

Friend(X,Y) => Friend(Y,X)

which means that if X is a friend of Y, then Y is a friend of X. This is not always true, but is probably true in many cases. Here's another example:

exists(Y) Friend(X,Y)

which means, for every X, there is at least one Y such that Friend(X,Y) is true. Don't worry about exact notation, as long as I can figure out what it means.

Some more examples:
Smokes(X) and Smokes(Y) => Friend(X,Y) or Friend(Y,X)
Friend(X,Y) and Smokes(X) => Smokes(Y)

Friend(X,Y) and !Smokes(X) => !Smokes(Y)
!Friend(X,Y) or !Friend(Y,Z) or Friend(X,Z).

Sometimes you have to be a little careful. Note that if I write “Friend(X,Y) and Smokes(X) => Smokes(Y)”, that does not say anything about what happens if X does not smoke, hence the need for another rule explicitly explaining what happens when “Friend(X,Y) and !Smokes(X)”. Also, of course it is not always true that a smoker’s friend smokes, but it might be more true than just 50%, so it can provide useful information/evidence to a system trying to figure out who smokes.

I would like as complete a theory as possible, but in order to help you focus your time, I would like you to pay particular attention to statements that involve the AdvisedBy() predicate, because that is probably the predicate I will be using for testing.

After you have written as many statements as you can, I would like you to go back and give each two numbers. The first (from 0-1) is the fraction of time that you think this statement is true. Don’t forget that a rule like “A => B” is true whenever A is false. The second number (from 0-1) is sort of a +/- which tells your confidence in first number. A value of 0 means it is absolutely right, and a value of 1 means you basically have no idea what it should be. For example:

0.8 0.4 Friend(X,Y) and Smokes(X) => Smokes(Y)

Means I think that maybe roughly 80% of smoker’s friends will smoke. But I’m not sure, it might be quite a bit more or less than 80%.

The rules themselves are more important to me than the numbers, so don’t stress over the numbers. Feel free to take whatever amount of time you want. Hopefully it will be a bit fun coming up with these statements. I’m hoping you can spend at least half an hour, but more time is certainly not discouraged!

B.2 Knowledge Base Produced by Volunteers

This is the knowledge base formed by merging the contributions from our four volunteers. Note, the division of the KB into sections is not part of the KB itself, but just for the purposes of making this appendix easier to understand. See Section B.1 for a list of the predicates and their argument types.

B.2.1 Miscellaneous

$$\texttt{TaughtBy}(\texttt{C},\texttt{P},\texttt{Q}) \wedge \texttt{CourseLevel}(\texttt{C},\texttt{level\_500}) \Rightarrow \texttt{Professor}(\texttt{P})$$

$$\texttt{TaughtBy}(\texttt{C},\texttt{P},\texttt{Q}) \wedge \texttt{Student}(\texttt{P}) \Rightarrow \neg\texttt{CourseLevel}(\texttt{C},\texttt{level\_500})$$

$$\texttt{TaughtBy}(\texttt{C},\texttt{P},\texttt{Q}) \wedge \texttt{Student}(\texttt{P}) \Rightarrow \neg\texttt{Phase}(\texttt{P},\texttt{pre\_Quals})$$

$$\texttt{TaughtBy}(\texttt{C},\texttt{P},\texttt{Q}) \wedge \texttt{Student}(\texttt{P}) \Rightarrow \neg\texttt{YearsInProgram}(\texttt{P},\texttt{year\_1})$$

$$\texttt{TaughtBy}(\texttt{C},\texttt{X},\texttt{Q}) \Rightarrow \texttt{Professor}(\texttt{X})$$

$$\texttt{TA}(\texttt{C},\texttt{X},\texttt{Q}) \Rightarrow \texttt{Student}(\texttt{X})$$

$$\texttt{TempAdvisedBy}(\texttt{P},\texttt{S}) \Rightarrow \texttt{Professor}(\texttt{P})$$

$$\texttt{TempAdvisedBy}(\texttt{P},\texttt{S}) \Rightarrow \texttt{Student}(\texttt{S})$$

$$\texttt{TempAdvisedBy}(\texttt{P},\texttt{S}) \Rightarrow \texttt{Position}(\texttt{P},\texttt{faculty})$$

$$\texttt{TempAdvisedBy}(\texttt{P},\texttt{S}) \Rightarrow \texttt{Phase}(\texttt{S},\texttt{pre\_Quals})$$

$$\texttt{TempAdvisedBy}(\texttt{P},\texttt{S}) \Rightarrow \texttt{YearsInProgram}(\texttt{P},\texttt{year\_1}) \vee \texttt{YearsInProgram}(\texttt{P},\texttt{year\_2})$$

$$\texttt{TA}(\texttt{C},\texttt{P},\texttt{Q}) \Rightarrow \texttt{Student}(\texttt{P})$$

$$\texttt{TaughtBy}(\texttt{C},\texttt{P},\texttt{Q}) \wedge \texttt{CourseLevel}(\texttt{C},\texttt{level\_500}) \wedge \texttt{TA}(\texttt{C},\texttt{S},\texttt{Q}) \Rightarrow$$
$$\texttt{AdvisedBy}(\texttt{S},\texttt{P}) \vee \texttt{TempAdvisedBy}(\texttt{S},\texttt{P})$$

$$\texttt{AdvisedBy}(\texttt{P},\texttt{S}) \Rightarrow \texttt{Student}(\texttt{S})$$

$$\texttt{AdvisedBy}(\texttt{P},\texttt{S}) \Rightarrow \texttt{Professor}(\texttt{P})$$

$$\texttt{AdvisedBy}(\texttt{P},\texttt{S}) \Rightarrow \neg\texttt{YearsInProgram}(\texttt{P},\texttt{year\_1})$$

$$\texttt{Publication}(\texttt{P},\texttt{X}) \wedge \texttt{Publication}(\texttt{P},\texttt{Y}) \wedge \texttt{Student}(\texttt{X}) \wedge \neg\texttt{Student}(\texttt{Y}) \Rightarrow \texttt{Professor}(\texttt{Y})$$

$$\texttt{Publication}(\texttt{P},\texttt{X}) \wedge \texttt{Publication}(\texttt{P},\texttt{Y}) \wedge \texttt{Student}(\texttt{X}) \wedge \neg\texttt{Student}(\texttt{Y}) \Rightarrow$$
$$\texttt{AdvisedBy}(\texttt{X},\texttt{Y}) \vee \texttt{TempAdvisedBy}(\texttt{X},\texttt{Y})$$

$$\texttt{Student}(\texttt{X}) \Rightarrow \neg\texttt{Professor}(\texttt{X})$$

$$\texttt{Professor}(\texttt{X}) \Rightarrow \neg\texttt{Student}(\texttt{Y})$$

$$\texttt{Student}(\texttt{X}) \Rightarrow \texttt{AdvisedBy}(\texttt{X},\texttt{Y}) \vee \texttt{TempAdvisedBy}(\texttt{X},\texttt{Y})$$

$\texttt{Professor}(\texttt{P}) \wedge \texttt{Position}(\texttt{P}, \texttt{faculty}) \Rightarrow \texttt{TaughtBy}(\texttt{C}, \texttt{P}, \texttt{Q})$

$\texttt{Phase}(\texttt{S}, \texttt{post\_Quals}) \Rightarrow \neg\texttt{YearsInProgram}(\texttt{year\_1})$

$\texttt{Phase}(\texttt{S}, \texttt{pre\_Quals}) \Rightarrow \neg\texttt{Phase}(\texttt{S}, \texttt{post\_Quals})$

$\texttt{Phase}(\texttt{S}, \texttt{pre\_Quals}) \Rightarrow \neg\texttt{Phase}(\texttt{S}, \texttt{post\_Generals})$

$\texttt{Phase}(\texttt{S}, \texttt{post\_Quals}) \Rightarrow \neg\texttt{Phase}(\texttt{S}, \texttt{pre\_Quals})$

$\texttt{Phase}(\texttt{S}, \texttt{post\_Quals}) \Rightarrow \neg\texttt{Phase}(\texttt{S}, \texttt{post\_Generals})$

$\texttt{Phase}(\texttt{S}, \texttt{post\_Generals}) \Rightarrow \neg\texttt{Phase}(\texttt{S}, \texttt{pre\_Quals})$

$\texttt{Phase}(\texttt{S}, \texttt{post\_Generals}) \Rightarrow \neg\texttt{Phase}(\texttt{S}, \texttt{post\_Quals})$

$\texttt{Professor}(\texttt{P}) \Rightarrow \texttt{Position}(\texttt{P}, \texttt{faculty}) \vee \texttt{Position}(\texttt{P}, \texttt{faculty\_affiliate}) \vee$
$\texttt{Position}(\texttt{P}, \texttt{faculty\_adjunct}) \vee \texttt{Position}(\texttt{P}, \texttt{faculty\_emeritus}) \vee$
$\texttt{Position}(\texttt{P}, \texttt{faculty\_visiting})$

$\texttt{Position}(\texttt{P}, \texttt{faculty\_visiting}) \Rightarrow \neg\texttt{AdvisedBy}(\texttt{S}, \texttt{P})$

$\texttt{Professor}(\texttt{X}) \wedge \texttt{Position}(\texttt{X}, \texttt{faculty}) \Rightarrow \texttt{AdvisedBy}(\texttt{S}, \texttt{X}) \vee \texttt{TempAdvisedBy}(\texttt{S}, \texttt{X})$

$\texttt{Student}(\texttt{P}) \wedge \neg\texttt{YearsInProgram}(\texttt{year\_1}) \Rightarrow \texttt{TA}(\texttt{C}, \texttt{P}, \texttt{Q})$

$\exists\texttt{Y}\ \texttt{TaughtBy}(\texttt{C}, \texttt{X}, \texttt{Q}) \Rightarrow \texttt{TA}(\texttt{C}, \texttt{Y}, \texttt{Q})$

$\exists\texttt{Y}\ \texttt{TA}(\texttt{C}, \texttt{X}, \texttt{Q}) \Rightarrow \texttt{TaughtBy}(\texttt{C}, \texttt{Y}, \texttt{Q})$

$\exists\texttt{P}\ \texttt{Phase}(\texttt{X}, \texttt{post\_Generals}) \Rightarrow \texttt{Publication}(\texttt{P}, \texttt{X})$

$\exists\texttt{Y}\ \texttt{Professor}(\texttt{X}) \Rightarrow \texttt{Position}(\texttt{X}, \texttt{Y})$

B.2.2 Advisement

$\texttt{AdvisedBy}(\texttt{X}, \texttt{Y}) \Rightarrow \texttt{Student}(\texttt{X})$

$\texttt{AdvisedBy}(\texttt{X}, \texttt{Y}) \Rightarrow \texttt{Professor}(\texttt{Y})$

$\texttt{TempAdvisedBy}(\texttt{X}, \texttt{Y}) \Rightarrow \texttt{Student}(\texttt{X})$

$\texttt{TempAdvisedBy}(\texttt{X}, \texttt{Y}) \Rightarrow \texttt{Professor}(\texttt{Y})$

$\texttt{Position}(\texttt{X}, \texttt{T}) \Rightarrow \texttt{Professor}(\texttt{X})$

$\texttt{TempAdvisedBy}(\texttt{X}, \texttt{Y}) \Rightarrow \neg\texttt{Position}(\texttt{X}, \texttt{faculty\_visiting})$

$\texttt{TempAdvisedBy}(\texttt{X}, \texttt{Y}) \wedge \neg\texttt{YearsInProgram}(\texttt{X}, \texttt{year\_1}) \Rightarrow \texttt{YearsInProgram}(\texttt{X}, \texttt{year\_2})$

$\texttt{TempAdvisedBy}(\texttt{X}, \texttt{Y}) \Rightarrow \texttt{Phase}(\texttt{X}, \texttt{pre\_Quals})$

$\exists \mathtt{Y}\ \mathtt{Student}(\mathtt{X}) \wedge \neg \mathtt{AdvisedBy}(\mathtt{X}, \mathtt{Y}) \Rightarrow \mathtt{TempAdvisedBy}(\mathtt{X}, \mathtt{Y})$

$\exists \mathtt{Y}\ \mathtt{Professor}(\mathtt{X}) \wedge \neg \mathtt{Position}(\mathtt{X}, \mathtt{faculty\_visiting}) \Rightarrow \mathtt{AdvisedBy}(\mathtt{Y}, \mathtt{X})$

B.2.3 Phases and positions

$\mathtt{Phase}(\mathtt{X}, \mathtt{Y}) \Rightarrow \mathtt{Student}(\mathtt{X})$

$\mathtt{Student}(\mathtt{X}) \Rightarrow \mathtt{Phase}(\mathtt{X}, \mathtt{pre\_Quals}) \vee \mathtt{Phase}(\mathtt{X}, \mathtt{post\_Quals}) \vee \mathtt{Phase}(\mathtt{X}, \mathtt{post\_Generals})$

$\mathtt{Phase}(\mathtt{X}, \mathtt{pre\_Quals}) \Rightarrow \neg \mathtt{Phase}(\mathtt{X}, \mathtt{post\_Quals})$

$\mathtt{Phase}(\mathtt{X}, \mathtt{post\_Quals}) \Rightarrow \neg \mathtt{Phase}(\mathtt{X}, \mathtt{pre\_Quals})$

$\mathtt{Phase}(\mathtt{X}, \mathtt{post\_Generals}) \Rightarrow \neg \mathtt{Phase}(\mathtt{X}, \mathtt{post\_Quals})$

$\mathtt{Position}(\mathtt{X}, \mathtt{Y}) \wedge \mathtt{Position}(\mathtt{X}, \mathtt{Z}) \Rightarrow \mathtt{SamePosition}(\mathtt{Y}, \mathtt{Z})$

B.2.4 Predicate constraints

$\neg \mathtt{AdvisedBy}(\mathtt{A}, \mathtt{A})$

$\neg \mathtt{TempAdvisedBy}(\mathtt{A}, \mathtt{A})$

$\mathtt{AdvisedBy}(\mathtt{A}, \mathtt{B}) \Rightarrow \neg \mathtt{AdvisedBy}(\mathtt{B}, \mathtt{A})$

$\mathtt{TempAdvisedBy}(\mathtt{A}, \mathtt{B}) \Rightarrow \neg \mathtt{TempAdvisedBy}(\mathtt{B}, \mathtt{A})$

$\mathtt{TempAdvisedBy}(\mathtt{S} : \mathtt{Person}, \mathtt{P} : \mathtt{Person}) \wedge \neg \mathtt{SamePerson}(\mathtt{P}, \mathtt{Q}) \Rightarrow$

$\quad \neg \mathtt{TempAdvisedBy}(\mathtt{S} : \mathtt{Person}, \mathtt{Q} : \mathtt{Person})$

$\mathtt{AdvisedBy}(\mathtt{S} : \mathtt{Person}, \mathtt{P} : \mathtt{Person}) \Rightarrow \neg \mathtt{TempAdvisedBy}(\mathtt{S} : \mathtt{Person}, \mathtt{Q} : \mathtt{Person})$

$\mathtt{TempAdvisedBy}(\mathtt{S} : \mathtt{Person}, \mathtt{P} : \mathtt{Person}) \Rightarrow \neg \mathtt{AdvisedBy}(\mathtt{S} : \mathtt{Person}, \mathtt{Q} : \mathtt{Person})$

B.2.5 Phase

$\mathtt{Phase}(\mathtt{S}, \mathtt{pre\_Quals}) \Rightarrow \neg \mathtt{AdvisedBy}(\mathtt{S}, \mathtt{P})$

$\mathtt{Phase}(\mathtt{S}, \mathtt{post\_Quals}) \Rightarrow \neg \mathtt{TempAdvisedBy}(\mathtt{S}, \mathtt{P})$

$\mathtt{Phase}(\mathtt{S}, \mathtt{post\_Generals}) \Rightarrow \neg \mathtt{TempAdvisedBy}(\mathtt{S}, \mathtt{P})$

B.2.6 Teaching and AdvisedBy

$$\texttt{Phase}(\texttt{S}, \texttt{post\_Quals}) \wedge \texttt{TaughtBy}(\texttt{C}, \texttt{P}, \texttt{Q}) \wedge \texttt{TA}(\texttt{C}, \texttt{S}, \texttt{Q}) \wedge \neg\texttt{CourseLevel}(\texttt{C}, \texttt{level\_100}) \Rightarrow \texttt{AdvisedBy}(\texttt{S}, \texttt{P})$$

$$\texttt{Phase}(\texttt{S}, \texttt{post\_Quals}) \wedge \texttt{TaughtBy}(\texttt{C}, \texttt{P}, \texttt{Q}) \wedge \neg\texttt{TA}(\texttt{C}, \texttt{S}, \texttt{Q}) \wedge \neg\texttt{CourseLevel}(\texttt{C}, \texttt{level\_100}) \Rightarrow \neg\texttt{AdvisedBy}(\texttt{S}, \texttt{P})$$

$$\texttt{Phase}(\texttt{S}, \texttt{post\_Quals}) \wedge \neg\texttt{TaughtBy}(\texttt{C}, \texttt{P}, \texttt{Q}) \wedge \texttt{TA}(\texttt{C}, \texttt{S}, \texttt{Q}) \wedge \neg\texttt{CourseLevel}(\texttt{C}, \texttt{level\_100}) \Rightarrow \neg\texttt{AdvisedBy}(\texttt{S}, \texttt{P})$$

$$\texttt{Phase}(\texttt{S}, \texttt{post\_Generals}) \wedge \texttt{TaughtBy}(\texttt{C}, \texttt{P}, \texttt{Q}) \wedge \texttt{TA}(\texttt{C}, \texttt{S}, \texttt{Q}) \wedge \neg\texttt{CourseLevel}(\texttt{C}, \texttt{level\_100}) \Rightarrow \texttt{AdvisedBy}(\texttt{S}, \texttt{P})$$

$$\texttt{Phase}(\texttt{S}, \texttt{post\_Generals}) \wedge \texttt{TaughtBy}(\texttt{C}, \texttt{P}, \texttt{Q}) \wedge \neg\texttt{TA}(\texttt{C}, \texttt{S}, \texttt{Q}) \wedge \neg\texttt{CourseLevel}(\texttt{C}, \texttt{level\_100}) \Rightarrow \neg\texttt{AdvisedBy}(\texttt{S}, \texttt{P})$$

$$\texttt{Phase}(\texttt{S}, \texttt{post\_Generals}) \wedge \neg\texttt{TaughtBy}(\texttt{C}, \texttt{P}, \texttt{Q}) \wedge \texttt{TA}(\texttt{C}, \texttt{S}, \texttt{Q}) \wedge \neg\texttt{CourseLevel}(\texttt{C}, \texttt{level\_100}) \Rightarrow \neg\texttt{AdvisedBy}(\texttt{S}, \texttt{P})$$

B.2.7 Publication and AdvisedBy

$$\texttt{Publication}(\texttt{T} : \texttt{Title}, \texttt{A} : \texttt{Person}) \wedge \texttt{Publication}(\texttt{T}, \texttt{B}) \wedge \neg\texttt{SamePerson}(\texttt{A}, \texttt{B}) \Rightarrow \texttt{AdvisedBy}(\texttt{A}, \texttt{B}) \vee \texttt{AdvisedBy}(\texttt{B}, \texttt{A})$$

$$\texttt{Publication}(\texttt{T}, \texttt{A}) \wedge \texttt{Publication}(\texttt{T}, \texttt{B}) \wedge \neg\texttt{SamePerson}(\texttt{A}, \texttt{B}) \wedge \texttt{Professor}(\texttt{A}) \wedge \texttt{Student}(\texttt{B}) \Rightarrow \texttt{AdvisedBy}(\texttt{B}, \texttt{A})$$

$$\texttt{AdvisedBy}(\texttt{S}, \texttt{P}) \wedge \texttt{Publication}(\texttt{T}, \texttt{S}) \Rightarrow \texttt{Publication}(\texttt{T}, \texttt{P})$$

B.2.8 Classes of people

$$\texttt{TaughtBy}(\texttt{C} : \texttt{Course}, \texttt{P} : \texttt{Person}, \texttt{Q} : \texttt{Quarter}) \Rightarrow \texttt{Professor}(\texttt{P} : \texttt{Person})$$

$$\texttt{Position}(\texttt{P} : \texttt{Person}, \texttt{X} : \texttt{Position}) \Rightarrow \texttt{Professor}(\texttt{P} : \texttt{Person})$$

$$\texttt{AdvisedBy}(\texttt{S} : \texttt{Person}, \texttt{P} : \texttt{Person}) \Rightarrow \texttt{Student}(\texttt{S} : \texttt{Person})$$

$$\texttt{AdvisedBy}(\texttt{S} : \texttt{Person}, \texttt{P} : \texttt{Person}) \Rightarrow \texttt{Professor}(\texttt{P} : \texttt{Person})$$

$$\texttt{Phase}(\texttt{P} : \texttt{Person}, \texttt{X} : \texttt{Phase}) \Rightarrow \texttt{Student}(\texttt{P} : \texttt{Person})$$

$\texttt{TempAdvisedBy}(\texttt{S} : \texttt{Person}, \texttt{P} : \texttt{Person}) \Rightarrow \texttt{Student}(\texttt{S} : \texttt{Person})$

$\texttt{TempAdvisedBy}(\texttt{S} : \texttt{Person}, \texttt{P} : \texttt{Person}) \Rightarrow \texttt{Professor}(\texttt{P} : \texttt{Person})$

$\texttt{YearsInProgram}(\texttt{P} : \texttt{Person}, \texttt{X} : \texttt{Integer}) \Rightarrow \texttt{Student}(\texttt{P} : \texttt{Person})$

$\texttt{TA}(\texttt{C} : \texttt{Course}, \texttt{P} : \texttt{Person}, \texttt{Q} : \texttt{Quarter}) \Rightarrow \texttt{Student}(\texttt{P} : \texttt{Person})$

B.2.9 People belong to one class

$\neg\texttt{Student}(\texttt{P} : \texttt{Person}) \Rightarrow \texttt{Professor}(\texttt{P} : \texttt{Person})$

$\texttt{Student}(\texttt{P} : \texttt{Person}) \Rightarrow \neg\texttt{Professor}(\texttt{P} : \texttt{Person})$

B.2.10 Uniqueness constraints

$\texttt{Position}(\texttt{P}, \texttt{X}) \wedge \neg\texttt{SamePosition}(\texttt{X}, \texttt{Y}) \Rightarrow \neg\texttt{Position}(\texttt{P}, \texttt{Y})$

$\texttt{Phase}(\texttt{P} : \texttt{Person}, \texttt{X} : \texttt{Phase}) \wedge \neg\texttt{SamePhase}(\texttt{X}, \texttt{Y}) \Rightarrow \neg\texttt{Phase}(\texttt{P} : \texttt{Person}, \texttt{Y} : \texttt{Phase})$

$\texttt{YearsInProgram}(\texttt{P} : \texttt{Person}, \texttt{X} : \texttt{Integer}) \wedge \neg\texttt{SameInteger}(\texttt{X}, \texttt{Y}) \Rightarrow$

$\quad \neg\texttt{YearsInProgram}(\texttt{P} : \texttt{Person}, \texttt{Y} : \texttt{Integer})$

B.2.11 Generally true uniqueness constraints

$\texttt{TaughtBy}(\texttt{X} : \texttt{Course}, \texttt{P} : \texttt{Person}, \texttt{Q} : \texttt{Quarter}) \wedge \neg\texttt{SameCourse}(\texttt{X}, \texttt{Y}) \Rightarrow$

$\quad \neg\texttt{TaughtBy}(\texttt{Y} : \texttt{Course}, \texttt{P} : \texttt{Person}, \texttt{Q} : \texttt{Quarter})$

$\texttt{TaughtBy}(\texttt{C} : \texttt{Course}, \texttt{X} : \texttt{Person}, \texttt{Q} : \texttt{Quarter}) \wedge \neg\texttt{SamePerson}(\texttt{X}, \texttt{Y}) \Rightarrow$

$\quad \neg\texttt{TaughtBy}(\texttt{C} : \texttt{Course}, \texttt{Y} : \texttt{Person}, \texttt{Q} : \texttt{Quarter})$

$\texttt{TA}(\texttt{X} : \texttt{Course}, \texttt{P} : \texttt{Person}, \texttt{Q} : \texttt{Quarter}) \wedge \neg\texttt{SameCourse}(\texttt{X}, \texttt{Y}) \Rightarrow$

$\quad \neg\texttt{TA}(\texttt{Y} : \texttt{Course}, \texttt{P} : \texttt{Person}, \texttt{Q} : \texttt{Quarter})$

$\texttt{TA}(\texttt{C} : \texttt{Course}, \texttt{X} : \texttt{Person}, \texttt{Q} : \texttt{Quarter}) \wedge \neg\texttt{SamePerson}(\texttt{X}, \texttt{Y}) \Rightarrow$

$\quad \neg\texttt{TA}(\texttt{C} : \texttt{Course}, \texttt{Y} : \texttt{Person}, \texttt{Q} : \texttt{Quarter})$

B.3 Algorithm Parameter Settings Used in the Experiments

B.3.1 Parameters used in training

- Weights are trained using pseudo-likelihood (PL)
- We optimize the PL by using conjugate gradient search with line minimization
- The PL is considered to have converged if it improves by no more than 0.001 from one iteration to the next. We also stop if it has taken 200 iterations. In our experiments, It usually converged in about 100-200 iterations.
- We used a Gaussian prior with mean and standard deviation of 1.0. A uniform prior produced nearly identical results.
- For the first iteration, the weights are initialized at the mean of their prior (1.0).
- We used exact counting of the number of satisfying assignments to ground clauses.

B.3.2 Parameters used in testing

- Testing is done using Gibbs sampling
- We take 10000 samples (full passes over all variables) for our probability estimates
- The sampling is split in to 10 “chunks”. Each chunk is initialized from a random starting point, then optimized using MaxWalkSat to find a mode of the distribution.
- Each chunk also has a burnin of 100 samples. That is, we do 100 passes over the variables before starting to collect the 1000 samples for that particular chunk.

B.3.3 Parameters used for Bayesian networks

- Data was binned by splitting into 5 bins of equal probability.
- Smoothing was done by adding 1 count to each entry in the CPTs.

- The class variable was added as a Boolean attribute for the BN, which was then used for class probability estimation.
- Structure learning was limited to no more than 4 parents per node.
- Vfbn2, part of VFML, was used for training the Bayesian network.

B.3.4 Parameters used for naive Bayes model

- Data was binned by splitting into 5 bins of equal probability.
- Smoothing was done by adding 1 count to each entry in the $P(X_i|class)$ multinomial.

B.3.5 Parameters used for Claudien

- scope(local)
- min_accuracy(0.1)
- search(breadth)
- beam_size(5)
- max_real_time(7200) (2 hours)
- min_coverage(1)
- max_complexity(10)

B.3.6 Claudien language bias

The language bias is set up to enable arbitrary predicates in the positive and negative sides of the clauses that claudien induces. I limit the number of variables per clause to 3, and allow each predicate to appear up to 2 times as a positive literal and up to 2 times as a negative literal. The bias is aware of predicate typing, which helps it limit the variables. To minimize search, equality

predicates (SameXXX) were not used in CLAUDIEN, and this improved its results.Here is the exact language bias:

```
dlab_template('
0-len:[
pTaughtBy(1-1:[C4_0,C4_1,C4_2],1-1:[C1_0,C1_1,C1_2], 1-
1:[C6_0,C6_1,C6_2]),
pTaughtBy(1-1:[C4_0,C4_1,C4_2],1-1:[C1_0,C1_1,C1_2], 1-
1:[C6_0,C6_1,C6_2]),
pCourseLevel(1-1:[C4_0,C4_1,C4_2],1-1:[C7_0,C7_1,C7_2]),
pCourseLevel(1-1:[C4_0,C4_1,C4_2],1-1:[C7_0,C7_1,C7_2]),
pPosition(1-1:[C1_0,C1_1,C1_2],1-1:[C8_0,C8_1,C8_2]),
pPosition(1-1:[C1_0,C1_1,C1_2],1-1:[C8_0,C8_1,C8_2]),
pProjectMember(1-1:[C5_0,C5_1,C5_2],1-1:[C1_0,C1_1,C1_2]),
pProjectMember(1-1:[C5_0,C5_1,C5_2],1-1:[C1_0,C1_1,C1_2]),
pAdvisedBy(1-1:[C1_0,C1_1,C1_2],1-1:[C1_0,C1_1,C1_2]),
pAdvisedBy(1-1:[C1_0,C1_1,C1_2],1-1:[C1_0,C1_1,C1_2]),
pPhase(1-1:[C1_0,C1_1,C1_2],1-1:[C9_0,C9_1,C9_2]), pPhase(1-
1:[C1_0,C1_1,C1_2],1-1:[C9_0,C9_1,C9_2]), pTempAdvisedBy(1-
1:[C1_0,C1_1,C1_2],1-1:[C1_0,C1_1,C1_2]), pTempAdvisedBy(1-
1:[C1_0,C1_1,C1_2],1-1:[C1_0,C1_1,C1_2]), pYearsInProgram(1-
1:[C1_0,C1_1,C1_2],1-1:[C10_0,C10_1,C10_2]), pYearsInProgram(1-
1:[C1_0,C1_1,C1_2],1-1:[C10_0,C10_1,C10_2]), pTA(1-
1:[C4_0,C4_1,C4_2],1-1:[C1_0,C1_1,C1_2],1-1:[C6_0,C6_1,C6_2]),
pTA(1-1:[C4_0,C4_1,C4_2],1-1:[C1_0,C1_1,C1_2],1-1:[C6_0,C6_1,C6_2]),
pProfessor(1-1:[C1_0,C1_1,C1_2]), pProfessor(1-1:[C1_0,C1_1,C1_2]),
pStudent(1-1:[C1_0,C1_1,C1_2]), pStudent(1-1:[C1_0,C1_1,C1_2]),
pPublication(1-1:[C3_0,C3_1,C3_2],1-1:[C1_0,C1_1,C1_2]),
pPublication(1-1:[C3_0,C3_1,C3_2],1-1:[C1_0,C1_1,C1_2]) ]
<--
```

```
0-len:[
pTaughtBy(1-1:[C4_0,C4_1,C4_2],1-1:[C1_0,C1_1,C1_2],
1-1:[C6_0,C6_1,C6_2]),
pTaughtBy(1-1:[C4_0,C4_1,C4_2],1-1:[C1_0,C1_1,C1_2],
1-1:[C6_0,C6_1,C6_2]),
pCourseLevel(1-1:[C4_0,C4_1,C4_2],1-1:[C7_0,C7_1,C7_2]),
pCourseLevel(1-1:[C4_0,C4_1,C4_2],1-1:[C7_0,C7_1,C7_2]),
pPosition(1-1:[C1_0,C1_1,C1_2],1-1:[C8_0,C8_1,C8_2]),
pPosition(1-1:[C1_0,C1_1,C1_2],1-1:[C8_0,C8_1,C8_2]),
pProjectMember(1-1:[C5_0,C5_1,C5_2],1-1:[C1_0,C1_1,C1_2]),
pProjectMember(1-1:[C5_0,C5_1,C5_2],1-1:[C1_0,C1_1,C1_2]),
pAdvisedBy(1-1:[C1_0,C1_1,C1_2],1-1:[C1_0,C1_1,C1_2]),
pAdvisedBy(1-1:[C1_0,C1_1,C1_2],1-1:[C1_0,C1_1,C1_2]),
pPhase(1-1:[C1_0,C1_1,C1_2],1-1:[C9_0,C9_1,C9_2]), pPhase(1-
1:[C1_0,C1_1,C1_2],1-1:[C9_0,C9_1,C9_2]), pTempAdvisedBy(1-
1:[C1_0,C1_1,C1_2],1-1:[C1_0,C1_1,C1_2]), pTempAdvisedBy(1-
1:[C1_0,C1_1,C1_2],1-1:[C1_0,C1_1,C1_2]), pYearsInProgram(1-
1:[C1_0,C1_1,C1_2],1-1:[C10_0,C10_1,C10_2]), pYearsInProgram(1-
1:[C1_0,C1_1,C1_2],1-1:[C10_0,C10_1,C10_2]), pTA(1-
1:[C4_0,C4_1,C4_2],1-1:[C1_0,C1_1,C1_2],1-1:[C6_0,C6_1,C6_2]),
pTA(1-1:[C4_0,C4_1,C4_2],1-1:[C1_0,C1_1,C1_2],1-1:[C6_0,C6_1,C6_2]),
pProfessor(1-1:[C1_0,C1_1,C1_2]), pProfessor(1-1:[C1_0,C1_1,C1_2]),
pStudent(1-1:[C1_0,C1_1,C1_2]), pStudent(1-1:[C1_0,C1_1,C1_2]),
pPublication(1-1:[C3_0,C3_1,C3_2],1-1:[C1_0,C1_1,C1_2]),
pPublication(1-1:[C3_0,C3_1,C3_2],1-1:[C1_0,C1_1,C1_2]) ] ').
```

Appendix C

ADDENDUM TO CHAPTER 6 EXPERIMENTAL SECTION

In this appendix, I give the set of directions which were given to our nine computer experts for our experiments on structure in the printer domain (see Section 6.2.2):

Hi,

Thanks for helping me out with this task. The next page gives a brief introduction in what we are asking you to do. Here are the directions:

1) Completely read the instructions on this page right now so that you know roughly what you'll be doing.
2) Read the introduction below, and read through the example problem to make sure you understand what we are asking you to do
3) Read the description of how printing works in windows
4) Complete the table by listing the causes of each node (if you can't finish, we understand. Please try to complete as much as possible, but we don't ask you to spend more than 45 minutes filling out this table). YOU MAY NOT COME BACK TO THIS STEP ONCE YOU HAVE DONE STEP 5!
5) Open the envelope. Inside is a list of all the "correct answers". Try to learn the causes of each variable by understanding the reason behind having those causes, not just using rote memorization. I have tried to help by giving the reasons (as far as I can tell) behind each set of causes.
6) Put the answers back in the envelope. Fill out the second table of causes.
7) Put it all in the envelope and in my mailbox in Sieg.

If you have any questions, just email me. There are no time limits on any part, so feel free to take as much time as you like. We understand that the task is hard, and confusing, so don't stress over it more than necessary. However, we do hope you will give it a full effort.

Thanks,
Matt Richardson
mattr@cs.washington.edu

This page describes the Windows 95 printer domain task. Please feel free to consult any source of knowledge (e.g. the internet, your printer manual, etc…) in performing the task. In the table below you'll find a list of variables (such as `DskLocal`), and a description of what each variable means ("Is there enough local disk space") (they are true/false valued variables). There are a number of mostly observation type variables, a number of intermediate variables, and a few final result variables. Your job is to write down which variables *directly influence* which other variables.

Imagine drawing arrows between the variables. An arrow from variable A to variable B would mean that A directly affects B. In other words, if you know the value of A, it helps you know the value of B. Try to consider the arrows a causal direction. That is, you would have a arrow from "smoking" to "cancer" since smoking causes cancer, not the other way around. If you drew arrows between all of the variables, you would have a list of "causes" of each variable. That is, the causes of a variable would be those that point to that variable ('smoking' is a cause of 'cancer'). Again, the causes of a variable X would be any variable which directly influences X.

You will be given a list of variables, and a blank area next to each one to list their "causes". Note that **any** direct influence should be considered a cause. For example, suppose A affects C, but only when B is true, then you would list both A and B as causes of C. As another example, suppose you determine a bunch of variables which affect whether connecting to a computer on the LAN would work, which you list as causes of *LANworks*, and you determine a bunch of variables which affect whether connecting to some computer outside of your LAN would work, which you list as causes of *WEBworks*. Suppose there is a variable *isLAN*, which is true if you are connecting on the LAN, and a variable *Works*, which is whether or not you can connect to some computer. Clearly, *Works* depends on *LANworks* when *isLAN* is true, and depends on *WEBworks* when *isLAN* is false. So, you would list all three as causes of *Works*. You would not list any of the causes of *LANworks* or causes of *WEBworks* as causes of *Works* because they are indirect, not direct causes.

There are some variables that are 'intermediate' and intended to reduce the number of causes each variable has. Please try to take advantage of these intermediate variables to simplify and reduce the number of causes when possible. For instance, if there were the variables "smoking", "dirty lungs", and "lung cancer", then there should be an arrow from smoking to dirty lungs, and from dirty lungs to lung cancer, but you don't need a line from smoking to lung cancer because it is an indirect cause of cancer, not a direct cause.

Some of the variables have no causes – they are things which would simply be observed and which are not affected by any of the other variables in the list. For those, please write an "X" in the blank.

Thank you so much for helping me out. For any questions, please email me!

P.S. Yes, I am asking you to write down a Bayesian Network (if you don't know what that is, don't worry about it).

Example Problem – the wet grass domain.
Here is an example of what you will be working on. You get a table such as this:

Variable	Short Description	Causes	Longer Description
Rain	Rain		Did it rain overnight?
Sprnklr	Water Sprinkler		Did the sprinkler come on overnight?
WtGrss	Wet Grass		Is the grass wet?
Clmsy	Clumsy Bob		Is Bob clumsy today?
Slip	Slip and Fall		Does Bob slip when walking across the lawn?

You are to fill out the 'causes' section, like this:

Variable	Short Description	Causes	Longer Description
Rain	Rain	*X*	Did it rain overnight?
Sprnklr	Water Sprinkler	*X*	Did the sprinkler come on overnight?
WtGrss	Wet Grass	*Rain, Sprnklr*	Is the grass wet?
Clmsy	Clumsy Bob	*X*	Is Bob clumsy today?
Slip	Slip and Fall	*WtGrss, clmsy*	Does Bob slip when walking across the lawn?

Rain, Sprnklr, and Clmsy are just facts – they are not influenced by any of the other variables. For instance, you cannot say "whether or not it rained last night depends on whether the grass is wet". However, note that whether the grass is wet **does** depend on whether it rained. So, WtGrss has Rain as a parent. It also depends on whether or not the sprinkler was running, so Sprnklr is a parent as well. Now, whether or not bob slips clearly depends on whether the grass is wet (WtGrss) and whether he is clumsy (clmsy), so those are causes. Not that whether Bob slips does depend on whether it rained and whether or not the sprinkler was on. But these are not **direct** influences. The direct cause of bob slipping is whether the grass is wet, so that is what we put as a parent instead.

Windows (and DOS) printing

Printers: Printers must be on, online, have enough toner and paper, etc.. in order to print properly. Usually printers have a number of indicator lights which tell whether there are problems with any of these.

Connection: There are two ways which the PC may be connected to the printer: Directly or through a network. A direct connection means the printer is connected directly with a cable to the printer port in the back of the computer. In this case, it is addressed by referring to the port it is connected to. Alternatively, the printer may be connected simply over the network. In this case, it is addressed by the "printer path" which describes where on the network the printer is.

Data: The way in which the data is sent to the printer depends on the operating system being used. In DOS, applications print directly to the port that the printer is on. So in that case, it is up to the application alone to know how to send the proper commands to the printer. Windows is much more complicated. In windows, each printer has a printer driver, which accepts input from applications and outputs it to the printer. The printer driver is part of the GDI (Graphical device interface). Windows applications send drawing commands to the GDI, which uses the printer driver to translate them into printer commands, which are then sent to the printer.

Spooling: To reduce the amount of time a person has to wait for the printer, windows uses "spooling". When an application prints and spooling is enabled, it does not get sent directly to the driver immediately. Instead, it is sent to the spool, and stored as EMF (Enhanced Metafile) data. Then, in the background, windows sends the EMF data to the driver to get translated into printer commands.

Postscript: Postscript is a language used by many printers to describe what they are to print. For postscript printers, the driver translates from the windows drawing commands into postscript commands. Being a full-fledged language, this translation, as well as the execution on the printer, often requires more memory (on the printer and/or on the computer) to process. Note that if the printer is postscript, that means the driver will be outputting postscript from windows, so "the printer is postscript" and "you are printing postscript stuff" are equivalent.

Fonts: There are two kinds of fonts: normal and true-type. True-type fonts are more rescalable, and look better than normal fonts. Further (for printers which support truetype fonts), truetype fonts are actually sent to the printer, so you are guaranteed that they will look as expected. When using standard fonts, it only works if the printer has the same font as the one you are using on the computer (I think…).

I have divided the variables into rough categories in order to help you see them as an organized set of variables. Note, variable causes can be variables in any category. For instance, it is easy to see that variables in the "Speed" section may depend on the printer, the connection, the data generation on the computer itself, etc… (and/or vice-versa). Here are all of the variables. The table on the following pages defines what each of them mean. Please read the entire table of variables and descriptions before you start listing causes, so you have an idea of what may be intermediate variables. You may find it useful to use scratch paper and draw lines between variables, cluster the variables, etc.. Good luck! (p.s. Note, many of these variables are spelled as if you simply remove the vowels from the correctly spelled word. Like "AvlblVrtlMmry" for "Available Virtual Memory".

Misc
REPEAT, PrtIcon, PrtQueue, PTROFFLINE, DS_NTOK, DS_LCLOK, AvlblVrtlMmry, FllCrrptdBffr

Printer
PrtOn, PrtPaper, TnrSpply, PrtStatPaper, PrtStatToner, PrtStatMem, PrtStatOff, PrtMem

Connection
PrtCbl, NetPrint, PrtPath, PrtPort, PrtSel, PC2PRT, NetOK, NtwrkCnfg, PrtMpTPth, CblPrtHrdwrOK

Drivers, spooling, data generation on the computer
DskLocal, PrtSpool, PrtDriver, DrvSet, DSApplctn, AppOK, DataFile, PrtFile, AppData, PrtDataOut, PrtThread, GDIOUT, EMFOK, GDIIN, DrvOK, LclOK

Postscript and Graphics
GrphcsRltdDrvrSttngs, PrtPScript, PSERRMEM, TstpsTxt, EPSGrphc, PSGRAPHIC, NnPSGrphc

Fonts
ScrnFntNtPrntrFnt, TrTypFnts, FntInstlltn, PrntrAccptsTrtyp, TTOK, NnTTOK

Garbled and/or Incomplete printing
PgOrnttnOK, PrntngArOK, CmpltPgPrntd, GrbldPS, IncmpltPS, GrbldOtpt, LclGrbld, NtGrbld

Speed
AppDtGnTm, HrglssDrtnAftrPrnt, DeskPrntSpd, PrntPrcssTm, PrtTimeOut, NtSpd

Problems
Problem1, Problem2, Problem3, Problem4, Problem5, Problem6

Printer Domain, Variable descriptions (****Before opening envelope****)
Fill out the "Causes" column to the best of your ability.

Variable	Short Description	Direct Causes	Longer Description
Misc			
REPEAT	Repeatable Problem		Is the problem repeatable (vs. sporadic)?
PrtIcon	Printer Icon		Does the windows printer icon look normal, or is it disabled (grayed out)
PrtQueue	Printer Queue		Is the printer queue not too long?
PTROFFLINE	Printer Driver Set Online		Is the printer driver set online?
DS_NTOK	DOS-NET OK		In DOS, when printing over the network, is the data sent to the printer okay?
DS_LCLOK	DOS-LOCAL OK		In DOS, when printing locally, is the data sent to the printer okay?
AvlblVrtlMmry	Printer Virtual Mem		Does the printer have enough virtual memory?
FllCrrptdBffr	Print Buffer		Is the print buffer okay, or is it full / corrupted?
Printer			
PrtOn	Printer On and Online		Is the printer on and ready (online)?
PrtPaper	Printer Paper Supply		Does the printer have paper?
TnrSpply	Toner Supply		Is there enough toner in the toner tray?
PrtStatPaper	Printer Status		Does the printer say no error, or does it have a paper jam / the paper bin is full or out of paper?
PrtStatToner	Printer Status		Does the printer say no error, or that it is low on toner?
PrtStatMem	Printer Status		Does the printer say no error, or that it is out of memory
PrtStatOff	Printer Status		Is the printer on/online or is the printer off / offline (the ONLINE LED is off)?
PrtMem	Printer Memory		Does the printer have more then 2 Megabytes of memory?
Connection			
PrtCbl	Local Printer Cable		Is the printer cable (running from the back of the computer to the printer) firmly connected?
NetPrint	Printing Locally		Is the printer connected by a local cable directly to the computer or connected via the network?
PrtPath	Net Printer		Is the networked printer path

	Pathname		correct?
PrtPort	Correct Local Port		For a local printer, is the correct port (parallel port) chosen?
PrtSel	Correct Printer Selected		Have you chosen to print to the correct printer?
PC2PRT	PC to PRT Transport		Is the path from computer to printer okay? (i.e. the printer gets the data)
NetOK	NET OK		Is the network working okay (in windows)?
NtwrkCnfg	Network Configuration		Is the network configuration correct?
PrtMpTPth	Port Mapping to Path		Is the mapping from port to printer path correct?
CblPrtHrdwrOK	Cable/Port Hardware		Are the cable and local printer port hardware okay?
Drivers, spooling, data generation on the computer			
DskLocal	Local Disk Space		Is there more than 2 Megabytes free on the local hard drive?
PrtSpool	Print Spooling		Is Printer Spooling enabled?
PrtDriver	Correct Driver		Is the correct printer driver installed?
DrvSet	Driver Configuration		Is the driver properly configured?
DSApplctn	Print Environment		Are you trying to print from DOS or Windows?
AppOK	Application		Is the application you are printing from working properly?
DataFile	Document		Is the datafile you loaded into the application okay?
PrtFile	Print to File		Can you successfully print to a file?
AppData	Application Data		Is the application outputting proper print data?
PrtDataOut	PrintDataOut		Is the data to be sent to the printer ok?
PrtThread	Port thread/Prt Proc OK		Is the printer spooling process ok?
GDIOUT	GDI Output OK		Is the windows driver output okay?
EMFOK	EMF OK		Is the Enhanced Metafile data sent to the driver okay?
GDIIN	GDI Input OK		Is the input to the driver okay?
DrvOK	Driver File Status		Is the driver installed okay?
LclOK	LOCAL OK		Is the local print connection working? (in windows)

Postscript and Graphics			
GrphcsRltdDrvrSttngs	Driver Config-Graphics		Are the graphics settings in the driver configuration correct?
PrtPScript	Postscript Printer		Are you using a Postscript printer?
PSERRMEM	PS Error Memory		Is it fine, or do you get a low memory error?
TstpsTxt	testps.txt Output		Does printing a small test.txt file to a postscript printer work?
EPSGrphc	EPS Graphic		Does your document contain normal images, or EPS (Encapsulated Postscript) images?
PSGRAPHIC	PS Graphic		Do graphics print properly (when using a postscript printer)?
NnPSGrphc	Non PS Graphic		Do graphics print properly on a non-postscript printer?
Fonts			
ScrnFntNtPrntrFnt	Screen Matches Printer		Do the fonts onscreen match the fonts that were printed?
TrTypFnts	True Type Fonts		Are you using True Type Fonts? (special fonts that can be scaled to any size)
FntInstlltn	Font Installation		Are fonts properly installed?
PrntrAccptsTrtyp	Printer Accepts Truetype		Does the printer accept True Type fonts?
TTOK	TT OK		Do TrueType fonts output correctly?
NnTTOK	Non TT OK		Do Non-TrueType fonts output correctly?
Garbled and/or Incomplete Printing			
PgOrnttnOK	Page Orientation		Did it print in the correct orientation (e.g. landscape vs. portrait)?
PrntngArOK	Printer Printing Area		Does the document fit in the area the printer can print?
CmpltPgPrntd	Non PS Complete		Does the page print completely? (if using a non-postscript printer)
GrbldPS	PS Not Garbled		Is the output okay (not garbled) when printing on a postscript printer?
IncmpltPS	PS Complete		When you try to print to postscript printer, does it print the entire document?
GrbldOtpt	Non PS Not Garbled		Is the output fine (not garbled) on a non-postscript printer?
LclGrbld	Local Garbled OK		Is the output okay (not garbled) when printing locally?
NtGrbld	Net Garbled OK		Is the output okay (not garbled) when printing over a network?

Speed			
AppDtGnTm	App Data Generation		Does the application take the right amount of time to send the data to the printer?
HrglssDrtnAftrPrnt	Hourglass Duration		Does the hourglass appear for only a short amount of time when you choose to print?
DeskPrntSpd	Desk Speed		Is the desktop computer fast enough?
PrntPrcssTm	Print Processing		Does the printer take the right amount of time to print?
PrtTimeOut	Printer Timeouts		Is the printer timeout setting too short?
NtSpd	Network printing speed		Is printing to a network printer fast enough?
Problems			
Problem1	Have Output		Did it output okay?
Problem2	Speed Normal		When you print, does it take a normal amount of time, or much longer than it should?
Problem3	Complete pages		Does it print full pages (vs. incomplete pages)?
Problem4	Graphics fine		Are the graphics fine, or are they distorted or incomplete?
Problem5	Fonts fine		Did fonts appear fine, or are some or all of the fonts missing or appear incorrect or distorted
Problem6	Output Looks normal		Is your output okay, or is it garbled?

The following table is to be filled in after opening the envelope and learning the causes of each variable. Remember to put the list of actual causes back in the envelope (e.g. don't use it while you are filling out the following table).

Printer Domain, Variable descriptions (**** **after opening envelope *****)
Please fill out the 'causes' column to the best of your abilities

Variable	Short Description	Direct Causes	Longer Description
Misc			
REPEAT	Repeatable Problem		Is the problem repeatable (vs. sporadic)?
PrtIcon	Printer Icon		Does the windows printer icon look normal, or is it disabled (grayed out)
PrtQueue	Printer Queue		Is the printer queue not too long?
PTROFFLINE	Printer Driver Set Online		Is the printer driver set online?
DS_NTOK	DOS-NET OK		In DOS, when printing over the network, is the data sent to the printer okay?
DS_LCLOK	DOS-LOCAL OK		In DOS, when printing locally, is the data sent to the printer okay?
AvlblVrtlMmry	Printer Virtual Mem		Does the printer have enough virtual memory?
FllCrrptdBffr	Print Buffer		Is the print buffer okay, or is it full / corrupted?
Printer			
PrtOn	Printer On and Online		Is the printer on and ready (online)?
PrtPaper	Printer Paper Supply		Does the printer have paper?
TnrSpply	Toner Supply		Is there enough toner in the toner tray?
PrtStatPaper	Printer Status		Does the printer say no error, or does it have a paper jam / the paper bin is full or out of paper?
PrtStatToner	Printer Status		Does the printer say no error, or that it is low on toner?
PrtStatMem	Printer Status		Does the printer say no error, or that it is out of memory
PrtStatOff	Printer Status		Is the printer on/online or is the printer off / offline (the ONLINE LED is off)?
PrtMem	Printer Memory		Does the printer have more then 2 Megabytes of memory?
Connection			
PrtCbl	Local Printer Cable		Is the printer cable (running from the back of the computer to the printer) firmly connected?
NetPrint	Printing Locally		Is the printer connected by a local cable directly to the computer or connected via the network?
PrtPath	Net Printer		Is the networked printer path

	Pathname		correct?
PrtPort	Correct Local Port		For a local printer, is the correct port (parallel port) chosen?
PrtSel	Correct Printer Selected		Have you chosen to print to the correct printer?
PC2PRT	PC to PRT Transport		Is the path from computer to printer okay? (i.e. the printer gets the data)
NetOK	NET OK		Is the network working okay (in windows)?
NtwrkCnfg	Network Configuration		Is the network configuration correct?
PrtMpTPth	Port Mapping to Path		Is the mapping from port to printer path correct?
CblPrtHrdwrOK	Cable/Port Hardware		Are the cable and local printer port hardware okay?
Drivers, spooling, data generation on the computer			
DskLocal	Local Disk Space		Is there more than 2 Megabytes free on the local hard drive?
PrtSpool	Print Spooling		Is Printer Spooling enabled?
PrtDriver	Correct Driver		Is the correct printer driver installed?
DrvSet	Driver Configuration		Is the driver properly configured?
DSApplctn	Print Environment		Are you trying to print from DOS or Windows?
AppOK	Application		Is the application you are printing from working properly?
DataFile	Document		Is the datafile you loaded into the application okay?
PrtFile	Print to File		Can you successfully print to a file?
AppData	Application Data		Is the application outputting proper print data?
PrtDataOut	PrintDataOut		Is the data to be sent to the printer ok?
PrtThread	Port thread/Prt Proc OK		Is the printer spooling process ok?
GDIOUT	GDI Output OK		Is the windows driver output okay?
EMFOK	EMF OK		Is the Enhanced Metafile data sent to the driver okay?
GDIIN	GDI Input OK		Is the input to the driver okay?
DrvOK	Driver File Status		Is the driver installed okay?
LclOK	LOCAL OK		Is the local print connection working? (in windows)

Postscript and Graphics			
GrphcsRltdDrvrSttngs	Driver Config-Graphics		Are the graphics settings in the driver configuration correct?
PrtPScript	Postscript Printer		Are you using a Postscript printer?
PSERRMEM	PS Error Memory		Is it fine, or do you get a low memory error?
TstpsTxt	testps.txt Output		Does printing a small test.txt file to a postscript printer work?
EPSGrphc	EPS Graphic		Does your document contain normal images, or EPS (Encapsulated Postscript) images?
PSGRAPHIC	PS Graphic		Do graphics print properly (when using a postscript printer)?
NnPSGrphc	Non PS Graphic		Do graphics print properly on a non-postscript printer?
Fonts			
ScrnFntNtPrntrFnt	Screen Matches Printer		Do the fonts onscreen match the fonts that were printed?
TrTypFnts	True Type Fonts		Are you using True Type Fonts? (special fonts that can be scaled to any size)
FntInstlltn	Font Installation		Are fonts properly installed?
PrntrAccptsTrtyp	Printer Accepts Truetype		Does the printer accept True Type fonts?
TTOK	TT OK		Do TrueType fonts output correctly?
NnTTOK	Non TT OK		Do Non-TrueType fonts output correctly?
Garbled and/or Incomplete Printing			
PgOrnttnOK	Page Orientation		Did it print in the correct orientation (e.g. landscape vs. portrait)?
PrntngArOK	Printer Printing Area		Does the document fit in the area the printer can print?
CmpltPgPrntd	Non PS Complete		Does the page print completely? (if using a non-postscript printer)
GrbldPS	PS Not Garbled		Is the output okay (not garbled) when printing on a postscript printer?
IncmpltPS	PS Complete		When you try to print to postscript printer, does it print the entire document?
GrbldOtpt	Non PS Not Garbled		Is the output fine (not garbled) on a non-postscript printer?
LclGrbld	Local Garbled OK		Is the output okay (not garbled) when printing locally?
NtGrbld	Net Garbled OK		Is the output okay (not garbled) when printing over a network?

Speed			
AppDtGnTm	App Data Generation		Does the application take the right amount of time to send the data to the printer?
HrglssDrtnAftrPrnt	Hourglass Duration		Does the hourglass appear for only a short amount of time when you choose to print?
DeskPrntSpd	Desk Speed		Is the desktop computer fast enough?
PrntPrcssTm	Print Processing		Does the printer take the right amount of time to print?
PrtTimeOut	Printer Timeouts		Is the printer timeout setting too short?
NtSpd	Network printing speed		Is printing to a network printer fast enough?
Problems			
Problem1	Have Output		Did it output okay?
Problem2	Speed Normal		When you print, does it take a normal amount of time, or much longer than it should?
Problem3	Complete pages		Does it print full pages (vs. incomplete pages)?
Problem4	Graphics fine		Are the graphics fine, or are they distorted or incomplete?
Problem5	Fonts fine		Did fonts appear fine, or are some or all of the fonts missing or appear incorrect or distorted
Problem6	Output Looks normal		Is your output okay, or is it garbled?

Correct List of causes

Variable	Short Description	Causes	Why these causes
Misc			
REPEAT	Repeatable Problem	CblPrtHrdwrOK, NtwrkCnfg	The problem will be repeatable as long as the hardware and network isn't flaky
PrtIcon	Printer Icon	NtwrkCnfg, PTROFFLINE	Whether you can see the printer icon depends on making sure the printer is online and the network is properly configured
PrtQueue	Printer Queue	none	None of the variables listed here affect the length of the queue – it is simply an observation.
PTROFFLINE	Printer Driver Set Online	none	As with PrtQueue, this is simply a factual observation (obs).
DS_NTOK	DOS-NET OK	AppData, PrtPath, PrtMpTPth, NtwrkCnfg, PTROFFLINE	Same as windows (NetOK), but DOS applications print directly to the printer, so you need to guarantee that AppData is good as well (windows apps are checked via PrintDataOut being okay, in the PC2PRT variable)
DS_LCLOK	DOS-LOCAL OK	AppData, PrtCbl, PrtPort, CblPrtHrdwrOK	As with DS_NTOK, this is the same as LclOK but with AppData added
AvlblVrtlMmry	Printer Virtual Mem	PrtPScript	If you aren't printing postscript, you always have enough memory
FllCrrptdBffr	Print Buffer	none	Observation (obs)
Printer			
PrtOn	Printer On and Online	none	Obs
PrtPaper	Printer Paper Supply	none	Obs
TnrSpply	Toner Supply	none	Obs
PrtStatPaper	Printer Status	PrtPaper	printer reports paper error if it's out of paper
PrtStatToner	Printer Status	TnrSpply	printer reports toner error if it's our of toner
PrtStatMem	Printer Status	PrtMem	Printer reports memory error if it's low on memory
PrtStatOff	Printer Status	PrtOn	printer reports being on if the printer *is* on
PrtMem	Printer Memory	None	Obs

Connection			
PrtCbl	Local Printer Cable	none	Obs
NetPrint	Printing over Network	none	Obs
PrtPath	Net Printer Pathname	none	Obs
PrtPort	Correct Local Port	None	Obs
PrtSel	Correct Printer Selected	None	Obs
PC2PRT	PC to PRT Transport	NetPrint, PrtDataOut, NetOK, LclOK, DSApplctn, DS_NTOK, DS_LCLOK	Any of these factors can clearly prevent the printer from getting correct data. For instance, NetOK affects whether the path from computer to printer is fine, but only matters if you are using a network printer (NetPrint) in the first place
NetOK	NET OK	PrtPath, NtwrkCnfg, PTROFFLINE	As long as you are using the correct printer pathname and the network is configured properly, we consider that the network is ok. I don't know why PTROFFLINE is here, heh heh.
NtwrkCnfg	Network Configuration	none	Obs
PrtMpTPth	Port Mapping to Path	None	Obs
CblPrtHrdwrOK	Cable/Port Hardware	none	Obs
Drivers, spooling, data generation on the computer			
DskLocal	Local Disk Space	none	Simply an observation
PrtSpool	Print Spooling	none	Observation
PrtDriver	Correct Driver	none	Obs
DrvSet	Driver Configuration	none	Obs
DSApplctn	Print Environment	None	Obs
AppOK	Application	None	Obs
DataFile	Document	None	Obs
PrtFile	Print to File	PrtDataOut	As long as the data being sent to the printer is good, it should correctly write a file
AppData	Application Data	AppOK, DataFile	As long as the data and application are okay, the application should be outputting correct data
PrtDataOut	PrintDataOut	GDIOUT, PrtSel	GDI outputs the data to the printer, so it is good as long as GDI is good and the correct printer has been selected
PrtThread	Port thread/Prt Proc OK	none	This is just a factual variable that doesn't depend on anything

GDIOUT	GDI Output OK	PrtDriver, GDIIN, DrvSet, DrvOK	GDI will properly output data as long as it was given proper data (GDIIN), and the drivers are all set up properly
EMFOK	EMF OK	AppData, DskLocal, PrtThread	EMF will be stored okay (when spooling) as long as the application produced correct data, there is enough local disk space to store the spool file, and the background process which does the spooling is working properly
GDIIN	GDI Input OK	AppData, PrtSpool, EMFOK	The driver will get correct input if the application output correct data. It also needs EMF to be working properly if spooling is enabled (since spooling is done to EMF format)
DrvOK	Driver File Status	none	Obs
LclOK	LOCAL OK	PrtCbl, PrtPort, CblPrtHrdwrOK	Local connection is fine if you are using a good cable, the correct port, and the port hardware is working properly
Postscript and Graphics			
GrphcsRltdDrvrSttngs	Driver Config- Graphics	None	Obs
PrtPScript	Postscript Printer	none	Obs
PSERRMEM	PS Error Memory	PrtPScript, AvlblVrtlMmry	Only run out of memory if using a postscript printer and have little virtual memory
TstpsTxt	testps.txt Output	PrtPScript, AvlblVrtlMmry	same as above
EPSGrphc	EPS Graphic	None	Obs
PSGRAPHIC	PS Graphic	PrtMem, GrphcsRltdDrvrSttngs	To print graphics, it needs enough memory and for graphics related driver settings to be set properly.
NnPSGrphc	Non PS Graphic	PrtMem, GrphcsRltdDrvrSttngs, EPSGrphc	Printing graphics on a non-postscript printer only works if they are not postscript (EPS) images, and also that the graphics settings are correct, and the printer has enough memory to print images.
Fonts			
ScrnFntNtPrntrFnt	Screen Matches Printer	None	Obs

TrTypFnts	True Type Fonts	None	Obs
FntInstlltn	Font Installation	None	Obs
PrntrAccptsTrtyp	Printer Accepts Truetype	None	Obs
TTOK	TT OK	PrtMem, FntInstlltn, PrntrAccptsTrtyp	True-type fonts will work best when you have enough memory, they are installed properly, and the printer accepts them
NnTTOK	Non TT OK	PrtMem, ScrnFntNtPrntrFnt, FntInstlltn	Non truetype fonts will work properly unless the screen doesn't match the printer font, or the fonts are installed wrong (or too little memory)
Garbled and/or Incomplete printing			
PgOrnttnOK	Page Orientation	None	Obs
PrntngArOK	Printer Printing Area	None	Obs
CmpltPgPrntd	Non PS Complete	PrtMem, PgOrnttnOK, PrntngArOK	The page orientation must be correct or it'll be cropped, as with the selection of print area. Having too little memory may cause the printer to end early, resulting in an only partially printed page
GrbldPS	PS Not Garbled	GrbldOtpt, AvlblVrtlMmry	All the same factors as GrbldOtpt but now, for postscript, you also need enough computer memory
IncmpltPS	PS Complete	CmpltPgPrntd, AvlblVrtlMmry	All the same factors as CmpltPgPrntd, but now, for postscript, you also need enough computer memory
GrbldOtpt	Non PS Not Garbled	NetPrint, LclGrbld, NtGrbld	Simply chooses between LclGrbld and NtGrbld depending on whether you are doing network or local printing
LclGrbld	Local Garbled OK	AppData, PrtDriver, PrtMem, CblPrtHrdwrOK	Any of these factors - bad data from application, bad driver, not enough memory, and bad cable port hardware, can make the output garbled when printing locally
NtGrbld	Net Garbled OK	AppData, PrtDriver, PrtMem, NtwrkCnfg	same as above, but for network
Speed			
AppDtGnTm	App Data Generation	PrtSpool	When the print spool is on, the application always seems to be fast enough.

			I guess. I honestly don't get this one
HrglssDrtnAftrPrnt	Hourglass Duration	AppDtGnTm	If the application takes a long time to generate the data, then the hourglass will appear for a long time
DeskPrntSpd	Desk Speed	PrtMem, AppDtGnTm, PrntPrcssTm	If the application generates it fast enough,
PrntPrcssTm	Print Processing	PrtSpool	I have no idea
PrtTimeOut	Printer Timeouts	none	Obs
NtSpd	Net Speed	DeskPrntSpd, NtwrkCnfg, PrtQueue	This speed is okay if the standard desktop printing speed is fine, but also the network is configured properly and the printer queue is short.
Problems			
Problem1	Have Output	PrtOn, PrtPaper, PC2PRT, PrtMem, PrtTimeOut, FllCrrptdBffr, TnrSpply	Pretty much everything
Problem2	Speed Normal	NetPrint, DeskPrntSpd, NtSpd	Selects between checking if printing on the desktop is fast enough, and whether printing via the network is fast enough (depending on whether you are or are not printing over the network)
Problem3	Complete pages	CmpltPgPrntd, PrtPScript, IncmpltPS	Simply selects between CmpltPgPrntd and IncmpltPS depending on whether you are or are not printing postscript
Problem4	Graphics fine	NnPSGrphc, PrtPScrpt, PSGRAPHIC	Graphics are okay as long as NnPSGrphc is okay or PSGRAPHIC is okay, depending on whether you are printing postscript
Problem5	Fonts fine	TrTypFnts, TTOK, NnTTOK	Fonts are fine if TTOK or NnTTOK, depending on whether you are using true type fonts
Problem6	Output Looks normal	GrbldOtpt, PrtPScript, GrbldPS	If PrtPScript, then the output being garbled depends on GrbldPS. Otherwise, it depends on GrbldOtpt

Appendix D

PROOF OF THEOREM 7.2.1

Here we give a proof of Theorem 7.2.1. We are assuming $\diamondsuit$ is commutative and associative, $\circ$ is associative and distributes over $\diamondsuit$, and **T**, $\mathcal{T}$, **b**, and $\mathcal{B}$ are defined as in Section 7.1. Also, from Section 7.2, $(\mathbf{A}\bullet\mathbf{B})_{ij} = \diamondsuit(\forall k\text{: } \mathbf{A}_{ik}\circ\mathbf{B}_{kj})$.

We first prove that $\bullet$is associative. Let **X**=(**A**$\bullet$**B**)$\bullet$**C**. Then:

$\mathbf{X}_{ij}$	$= \diamondsuit(\forall k\text{: } \diamondsuit(\forall l\text{: } \mathbf{A}_{il}\circ\mathbf{B}_{lk})\circ \mathbf{C}_{kj})$	from the definition of $\bullet$
	$= \diamondsuit(\forall k\text{: } \diamondsuit(\forall l\text{: } \mathbf{A}_{il}\circ\mathbf{B}_{lk}\circ\mathbf{C}_{kj}))$	since $\circ$ distributes over $\diamondsuit$ and $\circ$ is associative
	$= \diamondsuit(\forall l\text{: } \diamondsuit(\forall k\text{: } \mathbf{A}_{il}\circ\mathbf{B}_{lk}\circ\mathbf{C}_{kj}))$	since $\diamondsuit$ is associative
	$= \diamondsuit(\forall l\text{: } \mathbf{A}_{il}\circ\ \diamondsuit(\forall k\text{: } \mathbf{B}_{lk}\circ\mathbf{C}_{kj}))$	since $\circ$ distributes over $\diamondsuit$
	$= \diamondsuit(\forall l\text{: } \mathbf{A}_{il}\circ (\mathbf{B}\bullet\mathbf{C})_{lj})$	by definition of $\bullet$

This implies that

X = **A**$\bullet$(**B**$\bullet$**C**) by definition of $\bullet$.

We have $\mathcal{B}^{(0)} = \mathbf{b}$ and $\mathcal{B}^{(n)} = \mathbf{T}\bullet\mathcal{B}^{(n-1)}$, so $\mathcal{B}^{(n)} = \mathbf{T}\bullet(\mathbf{T}\bullet(\ldots\bullet(\mathbf{T}\bullet\mathbf{b})))$. Since $\bullet$is associative,

$$\mathcal{B}^{(n)} = \mathbf{T}^n\bullet\mathbf{b} \tag{D.1}$$

(where $\mathbf{T}^n$ means $\mathbf{T}\bullet\mathbf{T}\bullet\mathbf{T}\ldots$ n times, and $\mathbf{T}^0$ is the identity matrix).

We have $\mathcal{T}^{(0)} = \mathbf{T}$ and $\mathcal{T}^{(n)} = \mathbf{T}\bullet\mathcal{T}^{(n-1)}$, so $\mathcal{T}^{(n)} = \mathbf{T}\bullet(\mathbf{T}\bullet(\ldots\bullet(\mathbf{T}\bullet\mathbf{T})))$. Hence,

$$\mathcal{T}^{(n)} = \mathbf{T}\bullet\mathbf{T}^n \tag{D.2}$$

Combining Equations D.1 and D.2,

$$\mathbf{T}\bullet\mathcal{B}^{(n)} = \mathcal{T}^{(n)}\bullet\mathbf{b} \tag{D.3}$$

Since we run until convergence, this is sufficient to show that $\mathbf{T}\bullet\mathcal{B} = \mathcal{T}\bullet\mathbf{b}$.

VITA

Matthew Richardson (who prefers to be called “Matt”) was born in 1975 in Idaho, but quickly moved to the booming town of Vancouver, WA (no, not the Vancouver in Canada) (yes, there is another Vancouver, in Washington). After enough time there, he moved to sunny California where he received a Bachelor of Science (with Honors) from the California Institute of Technology in 1997, and was frequently thought to have come from Canada. Having had enough of southern California to last him his lifetime, he fled back north to Seattle and began graduate studies in the Computer Science and Engineering department at the University of Washington. He received his M.S. in Computer Science from the University in 1999, and a Ph.D. in the summer of 2004. On a rather hot day in fact.

Matthew’s primary interests are in machine learning and artificial intelligence. He has coauthored over a dozen publications on web search, social networks, networks of trust, the Semantic Web, statistical machine learning, and other topics. He has been a PC member or reviewer for nine conferences and workshops, and is a recipient of the IBM Ph.D. Fellowship.