Title
Extracting and Querying Probabilistic Information in BayesStore

Author
Wang, Zhe

Publication Date
2011

Peer reviewed|Thesis/dissertation

Extracting and Querying Probabilistic Information in BayesStore

by

Zhe Wang

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

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Michael J. Franklin, Chair
Professor Minos Garofalakis
Professor Joseph M. Hellerstein
Professor Cari Kaufman

Fall 2011

Extracting and Querying Probabilistic Information in BayesStore

Zhe Wang

Abstract

Extracting and Querying Probabilistic Information in BayesStore

by

Zhe Wang

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Michael J. Franklin, Chair

During the past few years, the number of applications that need to process large-scale data has grown remarkably. The data driving these applications are often uncertain, as is the analysis, which often involves probabilistic models and statistical inference. Examples include sensor-based monitoring, information extraction, and online advertising. Such applications require probabilistic data analysis (PDA), which is a family of queries over data, uncertainties, and probabilistic models that involve relational operators from database literature, as well as inference operators from statistical machine learning (SML) literature. Prior to our work, probabilistic database research advocated an approach in which uncertainty is modeled by attaching probabilities to data items. However, such systems do not and cannot take advantage of the wealth of SML research, because they are unable to represent and reason the pervasive probabilistic correlations in the data.

In this thesis, we propose, build, and evaluate BAYESSTORE, a probabilistic database system that natively supports SML models and various inference algorithms to perform advanced data analysis. This marriage of database and SML technologies creates a declarative and efficient probabilistic processing framework for applications dealing with large-scale uncertain data. We use sensor-based monitoring and information extraction over text as the two driving applications. Sensor network applications generate noisy sensor readings, on top of which a first-order Bayesian network model is used to capture the probability distribution. Information extraction applications generate uncertain entities from text using linear-chain conditional random fields. We explore a variety of research challenges, including extending the relational data model with probabilistic data and statistical models, efficiently implementing statistical inference algorithms in a database, defining relational operators (e.g., select, project, join) over probabilistic data and models, developing joint optimization of inference operators and the relational algebra, and devising novel query execution plans. The experimental results show: (1) statistical inference algorithms over probabilistic models can be efficiently implemented in the set-oriented programming framework in databases; (2) optimizations for query-driven SML inference lead to orders-of-magnitude speed-up on large corpora; and (3) using in-database SML methods to extract and query probabilistic information can significantly improve answer quality.

I dedicate this Ph.D. thesis to my parents and my family.

Contents

List of Figures

List of Tables

Acknowledgements

My advisors Mike Franklin, Joe Hellerstein, and Minos Garofalakis are the best advisors I could have asked for. I feel very fortunate to be able to work six years with them, share their vision, observe their research styles and methods, and learn from their critiques. All of my advisors contributed countless hours of advice and guidance regarding research, writing, speaking, and life in general. I learnt a great deal from each of them, and I can only hope that I can pass on their wisdom, experience, and their passion and sincerity towards research to my students.

Mike taught me to be a visionary researcher. The most frequent questions he would ask are: what is the point and why it matters. Being a Ph.D. student, it is very easy to be buried in details, Mike always has very clear high-level pictures and urges his student to work only on the problems that matters and show results that has a point. In addition, Mike taught me to be very thorough to details. For example, results and claims in papers need to be carefully quantified—board claims are unacceptable. Mike's tireless editorial effort vastly improved the quality of this dissertation. Mike also taught me the importance of public speaking, networking, and collaboration with other researchers in the community. Lastly, Mike is a very good friend. He is always ready to entertain, and my Ph.D. life would be less fun without his laughs and jokes.

Joe is a constant and proactive supporter of me and my thesis research. In the past six years, Joe has always been available to discuss my ideas and no matter how much critique he gives, he would always end on an encouraging high note. Joe is amazingly easy to discuss ideas, and he always gives very insightful feedback. In addition, Joe always looks out for his students for opportunities either for post-graduate jobs or for application of the research ideas in industry.

As with Mike and Joe, Minos has been a fearless leader of the BAYESSTORE project from the start. Before he left for a professorial position in Greece, Minos spent a lot of time working with me to form and solidify many basic ideas in BAYESSTORE. After he returned to Greece, the BAYESSTORE project continues to be shaped by his technical knowledge, ideas and intuitions. Minos is very dedicated to the project and to helping me establish my Ph.D. career. Also as with Mike and Joe, I remember Minos to be tough at times, but mostly encouraging and caring.

The Berkeley database group, especially Eirinaios Michelakis, David Chu, Alexandra Meliou, Kuang Chen, Tyson Condie, and Shawn Jeffery, and the Berkeley AMP lab, especially Beth Trushkowsky, Yanpei Chen, Kristal Curtis, Fabian Wauthier, Tim Kraska, and Justin Ma, were a constant source of ideas, expertise, friendship, and support throughout my doctoral studies. Our many discussions made this a better thesis.

Thanks also go to other faculties at UC Berkeley, who provided me with invaluable advice, including Cari Kauffman, Christos Papadimitriou, Michael Jordan, Randy Katz, and Stuart Russell. In addition, I would like to thank the wonderful staff in the CS department, especially La Shana Polaris, who made great effort to keep us informed and happy. Special thanks to Sean Mcmahon, who helped me greatly during the filing of this thesis.

I have been blessed throughout my life with many great teachers and mentors. I am particularly indebted to Alon Helevy, Shivakumar Vaithyanathan, Ken Sevcik, Arno Jacobsen, Kelly Lyons, and Philip Bohannon.

Finally, I am especially grateful to my mom and dad. They took a lot of effort to raise and

educate me well. They have always been a great support for me to realize my dreams. I would also like to thank my husband and life partner, Milenko Petrovic, for his passion in excellence that inspired me, for his sense of humor, and for his patience and understanding. I am thankful that my family and friends kept me happy, healthy, and normal for providing me a life outside of Ph.D. and computer science.

My research was supported in part by the National Science Foundation under ITR/IIS grant 0722077 and 0803690, research funds from IBM and Intel, AMPLab founding sponsors Google and SAP, RADLab sponsors AmazonWeb Services, Cloudera, Huawei, IBM, Intel, Microsoft, NEC, NetApp, andVMWare, and the European Commission under FP7-PEOPLE-2009-RG-249217 (HeisenData).

Curriculum Vitæ

Zhe Wang

Education

2005	University of Toronto, Canada B.A.Sc. Computer Engineering
2010	University of California, Berkeley M.S., Electrical Engineering and Computer Science
2011	University of California, Berkeley Ph.D., Electrical Engineering and Computer Science

Personal

Born	May 3, 1982, Hangzhou, Zhejiang, China

Awards

UC Berkeley	Department of Electrical Engineering and Computer Sciences Stonebraker/Wong Fellowship 2009 Departmental Fellowship 2005.
University of Toronto	Department of Electrical and Computer Engineering McAllister Research Award 2004, University of Toronto Scholar 2002, 2004, Dean's Honor List 2002-2004, Admission Scholarship 2002.

Academic Experience

Fall 2005-Spring2011	Research Assistant, Database Group, UC Berkeley Proposed, developed and evaluated BAYESSTORE, a probabilistic database system for statistical machine learning models. Devised efficient ways to natively support graphical models and their inference algorithms. Invented algorithms to perform probabilistic querying and analysis over large-scale uncertain data and the associated models. Designed query execution strategies that optimize across relational and inference operators.
Summer 2003-Spring 2005	Undergraduate Researcher, Middleware Systems Research Group, University of Toronto, Canada Designed and developed a demo for "Scalable Location-Aware Publish Subscribe System" and presented it at IBM CASCON 2004 conference. Designed, implemented and evaluated algorithms for large-scale XPath and regular expression matching.

Industrial Experience

Summer 2008	Research Internship, Yahoo! Research. Santa Clara

Investigated techniques to compress and visualize lineage in the debug/feedback phase of PSOX, a web-scale information extraction (IE) system. Significantly improved data bandwidth of IE pipelines that involve both machine learning and database operations.

Spring 2008 - Spring 2010 Research Collaborator, IBM Almaden
Research Center Developed and evaluated estimators for the cost and the output size of text extractors, such as dictionaries and regular expressions. Designed and evaluated different document synopses for more accurate estimation of various statistics over text corpora.

Summer 2007 Software Engineer, Google Inc.
Explored the complex problem of scalable extraction of the metadata of the HTML tables from the entire Web. Developed statistical classifiers and rule-based detectors, which recovered millions of schemas from HTML tables.

Summer 2006 Research Internship, Intel Research Berkeley
Designed and formalized the data model and developed the algorithms for relational operators over probabilistic data. This work formed the core of the BayesStore system.

Summer 2003, 2004, 2005 Software Developer, DB2 UDB Compiler Group, IBM Toronto Lab
Worked with large and complex codebase of DB2 on AIX Unix. Developed and tested new functionalities for materialized tables. The code of this line item was shipped with DB2 8.4. Developed components for dbtop - a performance monitoring utility in IBM DB2.

Publications

Refereed Conferences

C7 Hybrid In-Database Inference for Declarative Information Extraction. Daisy Zhe Wang, Michael J. Franklin, Minos Garofalakis, and Joseph M. Hellerstein, Michael L. Wick. To Appear, Proceedings of ACM SIGMOD International Conference on Management of Data, 2011.

C6 Selectivity Estimation for Extraction Operators over Text Data. Daisy Zhe Wang, Long Wei, Yunyao Li, Frederick Reiss, and Shivakumar Vaithyanathan. To Appear, Proceedings of 27th IEEE International Conference on Data Engineering (ICDE), 2011. (Acceptance Rate: 19.8

C5 Querying Probabilistic Information Extraction. Daisy Zhe Wang, Michael J. Franklin, Minos Garofalakis, and Joseph M. Hellerstein. Proceedings of 36th Very Large Data Base Endowment (VLDB), 2010, Vol.3: p1057-1067. (Acceptance Rate: 25

C4 Probabilistic Declarative Information Extraction. Daisy Zhe Wang, Eirinaios Michelakis, Michael J. Franklin, Minos Garofalakis, and Joseph M. Hellerstein Proceedings of 26th IEEE International Conference on Data Engineering (ICDE), 2010 short paper: p173-176. (Acceptance Rate: 20

C3 BayesStore: Managing Large, Uncertain Data Repositories with Probabilistic Graphical Models. Daisy Zhe Wang, Eirinaios Michelakis, Minos N.

	Garofalakis, Joseph M. Hellerstein. Proceedings of 34th Very Large Data Base Endowment (VLDB), 2008, Vol1: p340-351. (Acceptance rate: 16.5
C2	WebTables: Exploring the Power of Tables on the Web. Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, Yang Zhang. Proceedings of 34th Very Large Data Base Endowment (VLDB), 2008, Vol1: p538-549. (Acceptance rate: 16.5
C1	Bonsai: Exploration and Cultivation of Machine Learning Models David Purdy, Daisy Zhe Wang Proceedings of Joint Statistical Meetings (JSM), 2008.
Refereed Workshops	
W3	Functional Dependency Generation and Applications in Pay-as-you-go Data Integration Systems. Daisy Zhe Wang, Luna Dong, Anish Das Sarma, Michael J. Franklin, Alon Halevy Proceedings of the ACM SIGMOD WebDB, 2009.
W2	Uncovering the Relational Web. Michael Cafarella, Alon Halevy, Yang Zhang, Daisy Zhe Wang, Eugene Wu Proceedings of the ACM SIGMOD WebDB, 2008.
W1	Granularity Conscious Modeling for Probabilistic Databases Eirinaios Michelakis, Daisy Zhe Wang, Minos N. Garofalakis, Joseph M. Hellerstein Proceedings of ICDM DUNE, 2007: p501-506.
In Submission, Non-Refereed and Technical Reports	
M3	Probabilistic Data Management for Pervasive Computing: The Data Furnace Project. Minos N. Garofalakis, Kurt P. Brown, Michael J. Franklin, Joseph M. Hellerstein, Daisy Zhe Wang, Eirinaios Michelakis, Liviu Tancau, Eugene Wu, Shawn R. Jeffery, Ryan Aipperspach. IEEE Data Engineering Bulletin, 29, No. 1: p57-63, 2006.
M2	Probabilistic Complex Event Triggering. Daisy Zhe Wang, Eirinaios Michelakis, Liviu Tancau. UC Berkeley Technical Report, 2006.
M1	RETriever: Fast String Matching for Publish Subscribe Systems. Daisy Zhe Wang, Milenko Petrovic, Arno H. Jacobsen. University of Toronto Technical Report, 2005.

Teaching and Advising Experience

Spring 2006	Teaching Assistant,CS186 Introduction to Database Systems, UC Berkeley Taught weekly tutorial sessions. Designed midterm and final exam questions. Designed and conducted hands-on labs to develop internal PostgreSQL database components.
Long Wei	B.S. Student, Sep. 2009 - June. 2010 Guided Long through the implementation and evaluation of various document synopses for text databases.
Michael Zhang	M.S. Student, Aug. 2009 - Jan. 2010 Supervised Michael through the implementation of a BayesStore demonstration.

Dwight Crow	B.S. Student, Summer 2009 Guided Dwight through the feasibility study of clustering millions of HTML table schemas into domains.

Invited Talks

”Querying Probabilistic Information Extraction”	CSAIL Seminar, MIT, November 17, 2010.
	Database Seminar, University of Toronto, January 5, 2010.
”Declarative Information Extraction in a Probabilistic Database System”	
	Info Lab Seminar, Stanford, May, 2009.
”BayesStore: Supporting Statistical Models in Probabilistic Databases”	
	Machine Learning Tea, Berkeley, May 2009.

Community Service

External Conference Reviewer	VLDB, SIGMOD, ICDE.
Member of Graduate Admissions Committee	UC Berkeley EECS, 2010.
Organizer of DB Seminar	UC Berkeley EECS, 2009-2010.
Member of Women in Computer Science and Engineering	UC Berkeley, 2005-present.
Founder of Women in Engineering	University of Toronto, 2004-2005.

Chapter 1

Introduction

As massive scale data acquisition and storage becomes increasingly affordable, large amounts of both structured and unstructured data are being collected. A new class of applications is using advanced probabilistic data analysis to extract useful patterns, insights and trends from the collected data with the goal of gaining important scientific insights and business advantages. Existing systems and technologies such as relational database systems and statistical machine learning packages, fall short in supporting the combination of efficient large-scale data processing and advanced probabilistic modeling and inference. A common industry practice for probabilistic data analysis loosely connects relational database and statistical packages through data import and export. However, this design suffers problems, including inefficiency and information loss (i.e., inaccuracy), in supporting probabilistic data analysis tasks. What is needed is an integrated system that can natively support probabilistic models, uncertainties in data, inference and relational query processing, and optimization over large-scale datasets.

1.1 Bigger Data and Deeper Analysis

An ever-growing number of Web, science and enterprise applications are generating large amounts of structured and unstructured data. The 2011 IDC (International Data Corporation) Digital Universe study predicts that overall data in the world will grow by 50 times by 2020 to 35 Zettabytes [1] [1].

Structured data are data organized using a certain identifiable structure (i.e., data model), such as those in relational databases and spreadsheets. Structured data used to be mainly generated by human input. However, the increasingly ubiquitous deployment of information sensing devices in the past decade is generating huge amounts of structured data, which dwarfs the amount generated by human input [1]. These information sensing devises include: physical sensors, where sensor readings are collected for inventory management and environmental monitoring applications; soft sensors, where software logs and click streams are collected for log analysis and online advertise-

[1] 1 Zettabyte = 10^{21} bytes $= 1$ million Petabytes

ment applications; and bio-medical instruments, where measurements and genome sequences are collected for health informatics and computational genomic applications.

Unstructured data, on the other hand, refers to data that do not have a pre-defined data model. Unstructured data consists of text as well as multimedia data, such as audio, video and pictures. The traditional forms of text data include corporate documents, news, emails, books, and academic papers. More recently, with the success and popularity of social network applications, such as Twitter, Facebook, Yelp, and Blogspot, unstructured text data is being generated faster than ever before. Moreover, the amount of unstructured multimedia data is also experiencing rapid growth due to Web applications such as YouTube and Flickr. The IDC study predicts that unstructured information will make up about 90 percent of all data created in the next 10 years.

In order to extract useful information, patterns, insights, and trends from both structured and unstructured data, advanced data analysis is used, which goes beyond computing simple statistics such as count, max, and sum and uses probabilistic models and methods from statistical machine learning (SML) literature. SML-based data analysis enables: (1) statistical inference and prediction based on probabilistic SML models; (2) parsing and understanding the semantics of unstructured data; and (3) probability theories to reason about uncertainty in a principled manner. The result data of the SML-based data analysis are inherently associated with uncertainties and probabilities. Uncertainties specify all possible values of the result data, whereas probabilities specify confidence over the possible values. Next, we give examples of such advanced SML-based data analysis.

The first example involves sensor networks, where environmental indicators such as temperature and humidity for a specific location over time can be inferred from the nearby sensor readings using SML models. The inferred values are inherently associated with uncertainties and probabilities. Another example is probabilistic text analysis applications, in which person names, events, and company names are extracted from news articles and user sentiments for certain products are inferred from reviews using SML models. The extracted entities and sentiments also naturally carry uncertainties and probabilities. As a further example, predictive analysis and risk analysis uses SML models to, for example, predict the path of a hurricane in atmospheric sciences or estimate the risk of a portfolio in the finance industry. The hurricane path and risk prediction is again associated with uncertainties and probabilities.

Many companies have gained a competitive edge using these advanced analyses over large amounts of data. For example, Netflix and Splunk gained their business advantages by analyzing structured data such as user ratings and software logs respectively. Twitter and Facebook, on the other hand, generate profit by inferring user preferences and interests from unstructured text. Google applies advanced analysis algorithms to both structured and unstructured data, supporting products such as Maps, News, Search and so on.

1.2 Two Motivating Applications

The system and the techniques developed in this thesis are driven by two applications that need to support advanced SML-based analysis over large amounts of data: (1) sensor networks and (2) text analysis.

1.2.1 Sensor Networks

Sensor networks consist of spatially distributed autonomous sensors that monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants [2]. Sensors cooperatively pass their data through the network to a main location. Sensor networks have been widely deployed to monitor environment such as air pollution and forest fires, structures such as buildings and bridges, and human movement such as smart home and elderly care.

While the sensor readings are deterministic, the values of the physical properties (e.g., light, temperature) remains uncertain due to missing sensor readings from low battery life and unavoidable measurement errors from signal interference. Thus, although we have the measurement of temperature at a certain time and location, the actual value of the temperature remains uncertain.

TinyOS [3] and TinyDB [4] were developed at UC Berkeley as an operating system and data management system for these small and ubiquitous data sensing devices. While these systems are efficient in collecting sensor readings, they do not natively handle the inherent missing values and erroneous readings. Previous work [5] showed that SML models can be used to interpolate and smooth the raw sensor readings.

1.2.2 Information Extraction (IE)

Information extraction (IE) is one type of text analysis that extracts entities, such as person names, companies and events, from the text of news articles, emails and tweets. This is done by tagging the tokens in the text with labels (e.g., location, event), which are structural information (i.e., metadata) over the text. In other words, the goal of IE is to automatically extract structured information from unstructured text. For example, information extraction tools, such as Stanford NER [6] and MALLET [7], can extract the organization "NFL", the event "Super Bowl", the locations "Meadowlands Stadium", "Big Apple", "New York", and the sport teams "Jets" and "Giants" from the following piece of text.

NFL owners voted to put the 2014 Super Bowl in the new $\$1.6$ billion Meadowlands Stadium in the Big Apple, home to the New York Jets and Giants.

State-of-the-art IE over text is performed using probabilistic models and probabilistic inference algorithms from the SML literature. A probabilistic model is learned from the training data, which contains human-generated text and labels. The probabilistic model encodes (1) a probability distribution of all possible extractions (i.e., label sequence) that could be given to a piece of text, and (2) the statistical correlation between the token and the corresponding label assigned to it. For example, if the word is "NFL", it is highly likely that it should be labeled as an organization. The model also encodes the correlation between adjacent tokens and labels. For example, given the following token "Bowl", it is more likely that "Super" is part of the event name "Super Bowl".

1.3 Probabilistic Data Analysis (PDA)

As described in previous sections, SML models have been used in many applications to extract useful information from structured and unstructured data to gain important scientific insights and business advantages. While SML-based analysis is a key part of the data analysis to support this new class of applications, SML packages lack a number of other important data analysis functionalities. First, database-like relational query processing is not supported in SML packages to select, project, join, and aggregate different parts of the structured data. Second, declarative languages are not supported to specify ad-hoc queries for exploratory data analysis—procedural code needs to be written in SML packages to specify data analysis tasks. Third, real-time query processing and optimization is not supported for interactive query-answering interface—offline batch processing is employed in SML packages and an analysis task usually take hours to complete.

The goal of this thesis is to support real-time interactive *probabilistic data analysis* using declarative SQL-like query interface based on those SML models. We define probabilistic data analysis (PDA) as a family of queries over data, uncertainties, and SML models that involve both relational operators from the database literature, and inference operators from the statistical machine learning (SML) literature. Based on SML models, PDA queries can be used to analyze structured and unstructured data, to perform inference based on probabilistic models, and to execute relational queries over data with uncertainties.

We define probabilistic data analysis to incorporate:

1. A declarative SQL-like interface to define ad-hoc interactive queries;
2. Efficient real-time query processing that involve:
   (a) Statistical inference operations based on SML models;
   (b) Relational operations over both deterministic and uncertain data;
3. Analysis over both structured and unstructured data.

For the IE example from the previous subsection, the inference queries in PDA would use probabilistic IE models to extract entities (e.g., person name, companies, and locations) from the text data. The extracted entities are not deterministic: rather, they are associated with uncertainties and probabilities. For example, the model may generate uncertainties by extracting "Apple" as a company entity or extracting "Big Apple" as a location entity with different probabilities from our example text. Let us assume that an *entity table* is a database relation that stores all the extracted entities from the text with their uncertainties. The relational queries in PDA support basic select-project-join and aggregation over the entity table. Because the extracted entities in the entity table are uncertain, the relational queries need to be able to handle data with uncertainties and probabilities.

As another example of PDA, sensor networks applications must constantly reason with volumes of noisy sensor readings using statistical inference to accomplish tasks like motion detection and human behavior modeling. Based on the SML models learned over past sensor readings, PDA

tools can be used to infer the underlying distribution of the true temperature values and perform relational selection, projection, join, and aggregation over these probabilistic distributions.

Other applications that require probabilistic data analysis include Web applications (e.g. social networks, recommendation systems), where click-streams are analyzed to model customer behaviors; log analysis, where software companies analyze huge volumes of software logs to predict and debug errors at run time; data integration systems, where schema mapping and entity resolution results are full of uncertainties; computational genetics, where genome sequence reads are stored with confidence values; and crowdsourcing, where the answers from the "crowd" contain various types of errors that need to be reasoned about.

1.4 Existing Systems and Technologies

In this section, we review existing systems and technologies for data analysis, including traditional databases and data warehouses, statistical machine learning and a common industry practice that combines the two to perform probabilistic data analysis. We discuss the strengths and weaknesses of each system or technology in supporting probabilistic data analysis applications and justify why a new approach is needed.

1.4.1 Databases and Data Warehouses

Traditionally, relational databases and data warehouses have been widely adopted to query and analyze large relational datasets. Such systems efficiently support (1) relational algebra to perform data filtering, join, and aggregation using technologies such as caching, indexing, optimization and parallelization; (2) online analytical processing (OLAP), which performs scalable aggregation over large multi-dimensional data using additional technologies such as materialized views; and (3) data mining toolkits for classification and clustering over large-scale relational data. Relational databases and data warehouses have achieved great success in querying and analyzing data for business, government and Web applications in the past 40 years.

Such systems, however, fall short in supporting the new class of applications, data types and analysis we described in Section 1.1. First, traditional database systems do not natively support text analysis over unstructured data, such as information extraction described in Section 1.3. Although most databases can store text data and can create inverted indices to support keyword search queries over the text, the more advanced analysis of extracting the structural information from the unstructured text and performing general relational queries over the extracted data is not supported.

Second, traditional database systems assume that all data are deterministic. They do not handle uncertainties or probability distributions either in data storage or in query processing. Although current research in probabilistic databases has been exploring ways to store uncertainties and probabilities within the relational data model, as we will discuss in Section 2.4, most approaches lead to inefficiency and lack of expressiveness in storing probabilistic distributions. Finally, traditional database systems do not natively support advanced analysis involving probabilistic modeling and statistical inference. Although there has been some research on how to enable certain statistical methods in relational databases, as we will describe in Section 2.5, the general problem remains

open, especially regarding efficient and scalable support of probabilistic models and inference in databases.

More recently, research on probabilistic databases is aimed to efficiently store and query over uncertain and probabilistic data. However, most early work in this area [8, 9] assumed independence between data values, making it very expensive or impossible to encode the probabilistic correlations that commonly exist, for example, between the tokens and labels in information extraction. A detailed review of probabilistic database literature can be found in Section 2.4.

1.4.2 Statistical Machine Learning (SML)

Statistical machine learning (SML) is a scientific discipline concerned with designing and developing algorithms that allow computers to learn and generalize from observed data that represent incomplete information about statistical phenomenon, in order to develop models that can make predictions on missing attributes or future data.

Many such models inherently encode a probability distribution over the possible values of the data. For example, based on sensor readings, machine learning models can be learned to capture the distribution of the true temperature values. When an inference operation is performed over data based on such probabilistic models, the results of the inference operation are associated with probability distributions. Probabilistic machine learning models have been successfully applied to many applications, including interpolation of missing values in sensor networks, extraction of entities, facts and relationships from text, and prediction of trends and patterns for online advertisement and log analysis.

Over the years, a number of SML packages have been developed to support both learning algorithms that generate probabilistic models from empirical data and inference algorithms that compute the top-k highest probability answers or probability distributions. Compared to database systems, these packages, including Rapid Miner, Weka, and Apache Mahout, can perform more advanced analysis such as text analysis over unstructured data and statistical inference over probabilistic models with uncertain data. Moreover, these packages are efficiently implemented using data structures common in statistics and probability theory, such as arrays and matrices. They also use procedural languages, such as Java and C++, which are natural for implementing sophisticated learning and inference algorithms, usually involving iterative and recursive procedures.

In Standalone SML packages, however, cannot fully support the requirements of PDA requirements identified in Section 1.3. This is because SML packages do not support important functionalities required for advanced data analysis: (1) relational query operations (e.g., select, project, join, aggregate) over data; (2) declarative languages to specify ad-hoc exploratory queries; and (3) real-time PDA query processing and optimization for interactive query-answering interface.

1.4.3 Loose Coupling of Databases and SML

A common industry practice to perform PDA uses machine learning packages to perform statistical inference over probabilistic models for both structured and unstructured data, and separately uses relational databases to store and query over the original data and the analysis results. Figure 1.1

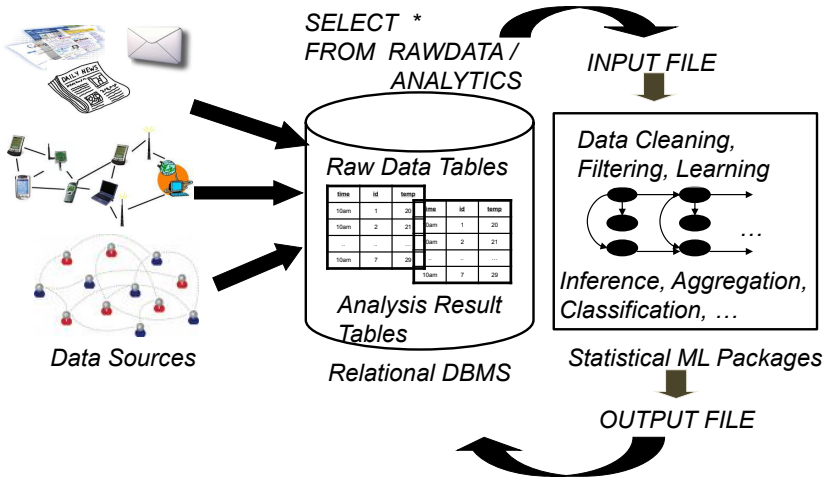


Figure 1.1. Illustration of the common practice for probabilistic data analysis.

shows the architecture of this common practice. Data from sources such as online news, sensor readings, and tweets are stored in a relational database. At the time of data analysis, the raw data are exported from the database into a file, organized into the required format, and fed into a machine learning package. Given a previously learned probabilistic model, a series of probabilistic data analysis tasks are performed over the new data inside the machine learning packages, including data cleaning, feature extraction, and statistical inference. Finally, the analysis results are imported back into the database for users to query.

There are many problems with this common practice for PDA.

The first problem is poor performance due to *exhaustive* data processing. In the common practice, user queries are issued on top of the database, whereas the probabilistic analysis is performed outside of the database. This setup requires the probabilistic analysis to be performed exhaustively (i.e., over all the data), because it is unaware of the user queries. For example, information extraction has to be performed over all possible entities for the entire corpus before loading into the database. This common practice results in a batch process for probabilistic analysis, which is inefficient and inflexible for ad-hoc queries. Moreover, because all the data analysis computations are done outside of the database (in SML packages), they do not take advantage of the facilities in the database for efficient data storage and retrieval, such as indexing, caching, and optimization.

The second problem is information loss. The common practice imports only the top-1 highest probability results as deterministic values into the relational database systems for users to query. This is because relational databases do not handle values with uncertainties or probabilities. However, the top-1 highest probability results generated from the machine learning models are not always correct. In fact, the accuracy of the state-of-the-art IE models can vary from 80% to the upper 90th percentile for different applications. This process prematurely discards all the uncertainties and probabilities inherent from the data analysis and in the analysis results, which may actually contain the correct answers that the users are looking for.

To continue the IE example from Section 1.2.2, assume that the model mistakenly extracts

"Apple" as a company entity with the highest probability, and "Big Apple" as a city entity with a lower probability. If we populate the entity table with only the top-1 extractions from this model, the following query

select all documents which contain a city entity "Big Apple"

would not return our example document and would possibly be missing many other correct answers as well.

If the database system could support the storage of and query processing over uncertainties and probability distributions over all possible extractions, the above query would result in a list of documents, including the document shown in the IE example in Section 1.2.2. This list of documents could be ranked in descending order of the probabilities of the extractions that contain a city entity "Big Apple". Although the returned extractions will include results that are not the highest probability, such results may well be the correct answers that are missing from those generated using the common practice.

Other problems with the common practice of loose coupling for PDA include: (1) it does not support online analysis queries using SML models; (2) it does not support a declarative query interface for SML-based analysis tasks; (3) it does not support indices over the data for SML-based analysis; and (4) it uses blackbox implementations of SML methods, which prevents many types of performance optimizations.

1.4.4 PDA Summary

Traditional database systems are unable to support PDA applications, because they lack the ability to handle uncertainties and probability distributions, and the ability to deal with data types other than structured relational data. On the other hand, SML packages lack the ability to efficiently scale up to large datasets, and the ability to handle interactive and declarative queries. The common industry practice for probabilistic data analysis loosely couples relational databases with SML packages to enable relational queries over the result of statistical inference with probabilistic models. Although such an approach can be used to implement probabilistic data analysis applications, it sacrifices efficiency and scalability by keeping probabilistic analysis outside of database as a batch process. It also sacrifices answer quality by allowing users to query only the most likely view of the analysis result distributions, prematurely discarding the uncertainties and probabilities inherent in the data and analysis, which may actually contain the correct answers for the user queries.

1.5 Our Approach: BAYESSTORE

In order to address the weaknesses of the current systems and technologies for probabilistic data analysis applications, in this thesis, we describe the design, implementation and evaluation of BAYESSTORE, an integrated probabilistic database system, which natively and efficiently supports probabilistic models, statistical inference, and relational queries over uncertain data and probability

distributions. The class of probabilistic models used in BAYESSTORE is known as Probabilistic Graphical Models, which we describe in detail in Section 2.1.

The main design guideline of BAYESSTORE is to bring probabilistic analysis models and methods, implemented in SML packages closer to the data in the database. This is done by implementing and integrating these computationally intensive analysis tasks into the data processing engine of the database. This approach allows probabilistic analysis to be performed more efficiently using database facilities, enables a deep integration and optimization between inference and relational operators in a single PDA task, and enables uncertainties and probabilities to be preserved and reasoned about during query processing.

The main features of BAYESSTORE are the following:

First, BAYESSTORE supports the storage of both structured relational data and unstructured text data. It also supports the storage of uncertain data through "incomplete tables", where the uncertain data are represented as missing values in the data table. The possible values of the uncertain data and the probability distributions over them are encoded in a probabilistic distribution.

Second, BAYESSTORE adopts a clean separation of the relational and probabilistic components, unlike most earlier work in probabilistic databases. Rather than attaching probabilities to uncertain data elements, BAYESSTORE maintains probability distributions as first-class entities in the database, separate from the data tables. This design has a number of benefits. First, it allows users to expose all probabilistic information as database objects that can be queried and manipulated by users and applications. Second, and perhaps most important, it allows users to leverage ideas from state-of-the-art SML in order to effectively represent, query, and manipulate probabilistic distributions.

Third, BAYESSTORE supports both relational and inference operators in processing and optimizing probabilistic data analysis queries. Query processing over uncertain data requires special algorithms to handle the associated uncertainties and probability distributions: in BAYESSTORE, relational operators are redefined to be applied to the uncertain data; and inference operators are implemented to perform statistical inference over uncertain data. Query optimization techniques are also developed for BAYESSTORE to optimize the runtime performance of PDA queries that involve inference operations.

To summarize, BAYESSTORE is a unified framework that combines the advanced probabilistic modeling and analysis power from statistical machine learning with the scalable data processing and management power of relational database systems to support real-time probabilistic data analysis over large-scale datasets.

1.6 Contributions

This dissertation contains five major contributions.

- **In-Database Representation for Probabilistic Data and Model (Chapter 3)**: We designed an efficient in-database representation for both deterministic and uncertain data, and for different types of data: structured or unstructured. We also developed an in-database repre-

sentation for probabilistic models encoding probabilistic distributions over uncertain data values.

- **Probabilistic Relational Operators (Chapter 4)**: We invented new algorithms for probabilistic relational operators, including select, project, and join over probabilistic models and data. These algorithms allow probability distributions to be queried, and allow uncertainties and probabilities to be populated through data processing pipelines in a principled manner.

- **In-Database Probabilistic Inference Operators (Chapter 5)**: We developed native in-database implementations for four inference algorithms over probabilistic graphical models, including Viterbi, sum-product, MCMC Metropolis Hastings, and MCMC Gibbs sampling. These implementations are shown (in Sections 5.3.2 and 5.4.2) to have comparable runtime to open source implementation of the same algorithms in procedural languages. This result shows that sophisticated statistical methods, involving recursive and iterative procedures, can be efficiently implemented in a set-oriented data processing framework like PostgreSQL. These native implementations also open up the opportunity for optimizations between the relational and the inference operators within a single probabilistic data analysis task.

- **Probabilistic Query Processing (Chapter 6)**: We crated query-driven techniques for executing PDA queries that involve both inference operators and relational operators. These techniques achieve a deep integration between the relational operators and the native implementation of the inference operators. Experiments (in Section 6.5) show that (1) querying over distributions rather than the top-1 highest-probability results can reduce false negatives (i.e., missing results) by as much as 80%; and (2) these query-driven techniques can achieve orders-of-magnitude speed-up on large-scale datasets compared to the batch process used in the common practice of loose coupling for PDA.

- **Query Rewrites and Optimizations (Chapter 7)**: Lastly, we developed a query optimization technique called "hybrid inference", which can choose among the different inference algorithms we implemented in BAYESSTORE. Within a single PDA query, and for each data item, hybrid inference picks an appropriate inference algorithm in terms of runtime and accuracy, based on the data, the model, and the query. An analysis (Section 7.7) over three datasets, including New York Times, Twitter and DBLP, shows that this optimization technique can achieve 5 to 10 times speed-ups compared to standard algorithms used in the SML literature.

1.7 Summary

A growing number of applications require advanced probabilistic data analysis over large amounts of structured or unstructured data to extract insights and predict trends in business, social science, and in medical domains, among others. In this chapter, we described how this new type of probabilistic data analysis is applied to reason about noisy sensor readings and to extract structural information from unstructured text—the two motivating applications of this thesis. We outlined the existing technologies and the common practice of loose coupling these technologies to perform PDA as well as their weaknesses. Finally, we summarized the BAYESSTORE, an integrated

data analysis system supporting both relational and inference queries over probabilistic models and uncertain data for PDA. We also explained how such an integrated system is more efficient in runtime, enabling real-time interactive analysis, and by supporting uncertain data processing provides better answers than those generated using the loose coupling approach.

The remainder of this dissertation justifies these claims more completely. In the next chapter, we review the details of probabilistic graphical models and their inference operations for probabilistic data analysis, and survey the related work in the area of probabilistic databases. In Chapter 3, we give an overview of the BAYESSTORE system, and describe its new data model, extended from the relational data model. Chapters 4 and 5 discuss the algorithms and in-database implementations of probabilistic relational operators and inference operators. Chapters 6 and 7 discuss the query processing algorithms and optimization techniques for probabilistic queries, involving both relational and inference operators. In Chapter 8, we discuss the remaining open issues and sketch directions for future work. Chapter 9 presents the conclusions of this dissertation.

Chapter 2

Background

In this chapter, we first introduce the class of probabilistic SML models that we use in this thesis—probabilistic graphical models (PGMs). We describe in detail two types of PGMs that are used in the two motivating applications of this thesis: Bayesian networks for sensor networks applications and conditional random fields for IE applications. We outline the basic inference algorithms over the two types of PGMs that are needed for PDA queries. We then survey related work on probabilistic databases, in-database statistical models, and statistical relational learning. We describe the differences between these related work and the work in this thesis and justify why this thesis work is needed to support the PDA requirements identified in Section 1.3.

2.1 Probabilistic Graphical Models (PGMs)

In this thesis, we focus on one class of probabilistic models in the SML literature to perform probabilistic data analysis, namely probabilistic graphical models (PGMs). In this section, we give an overview of PGMs in general and a primer on the two types of PGMs that drive this thesis work: Bayesian networks and conditional random fields.

2.1.1 Overview

Probabilistic graphical models (PGMs) [10, 11, 12, 13, 14] have become an extremely popular tool for modeling uncertainty. They provide a principled way to deal with uncertainty through the use of probability theory, and an effective approach for coping with complexity through the use of graph theory.

At a high level, the goal is to efficiently represent a joint distribution P over some set of random variables $\mathfrak{X} = \{X_1, ..., X_n\}$. Even in the simplest case where these variables are binary-valued, a joint distribution requires the specification of 2^n numbers – the probability of the 2^n different assignments of values $x_1, ..., x_n$. However, it is often the case that there is some structure in the distribution that allows us to factor the representation of the distribution into modular components.

The structure that graphical models exploits is the independence properties that exist in many real-world applications. The independence properties in the distribution can be used to represent such high-dimensional distributions much more compactly.

A PGM is a graph $G = (V, E)$, which consists of a set of random variables (the nodes of the graph V), associated to the entities from the problem domain, and a set of conditionally independent relationships among them, encoded by the edge set E. Probabilistic graphical models provide a general-purpose modeling language for exploiting conditional independence structure between random variables. Inference over PGMs provides us with the mechanisms for computing the top-k highest probability values or the distribution of a subset of random variables in V.

Two major families of graphical models dominate the related literature. Bayesian Networks (also called belief networks or causal networks) are defined over directed graphs; Markov Random Fields (also called Markov networks) follow an undirected graphical representation. The independence semantics of the former state that a node is conditionally independent of any other node in the network, when conditioned on its parents (the nodes that point to it). In the case of Markov Random Fields, where the edge directionality is dropped, the independence semantics allow two sets of RVs to be regarded as independent when conditioned on a third set of RVs that completely separates the two.

PGMs have been successfully applied to numerous applications in many domains [10, 15, 12]. Applications of graphical models include information extraction, speech recognition, computer vision, modeling of noisy mobile sensor data, gene finding, and diagnosis of diseases. Probabilistic data analysis in different applications applies different PGMs. In the following sections, we describe in detail two types of PGMs—Bayesian networks and conditional random fields.

2.1.2 Bayesian Networks

In this section, we introduce general Bayesian networks, which are a set of directed probabilistic graphical models. In Chapter 3.3.2, we describe a Bayesian network variant we use for sensor network applications.

Over the past 20 years, *Bayesian networks* (BNs) have emerged as a widely-used statistical machine learning (SML) model to efficiently represent and reason about multi-variate probabilistic distributions [16]. A BN is a directed acyclic graph (DAG) $G = (V, E)$ with nodes V representing a set of RVs $\{X_1, X_2, \ldots, X_{|V|}\}$, and edges E denoting conditional dependence relationships between RVs. In a nutshell, the structure of the BN encodes the assertion that *each node is conditionally independent of its non-descendants, given its parents*. This allows us to model the probability distribution of a RV X_i, *locally*, as a *conditional probability table* (CPT) or *factor* $\mathsf{Pr}[X_i|Pa(X_i)]$ that specifies how X_i depends probabilistically on the values of its parent nodes $Pa(X_i)$. It is not difficult to see that the overall joint probability distribution can be factored into the product of all local factors in the DAG; that is, $\mathsf{Pr}[X_1, ..., X_{|V|}] = \prod_{i=1}^{|V|} \mathsf{Pr}[X_i|Pa(X_i)]$. Given that the dimensionality of local correlation models is typically much smaller than $|V|$, this factored BN representation can be orders of magnitude more efficient than a straightforward $|V|$-dimensional probability distribution.

Bayesian networks can be used to model the probability distribution over noisy sensor readings.

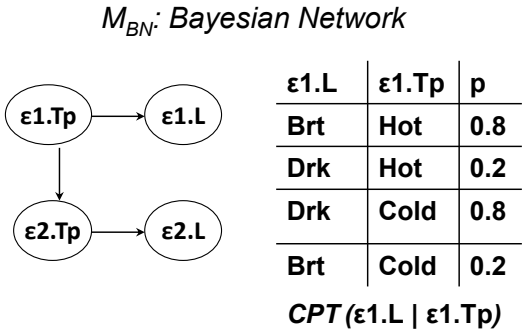


Figure 2.1. An example Bayesian network of temperature (Tp) and light (L) readings from two sensors $\{\varepsilon_1, \varepsilon_2\}$; and the CPT associated with RV $\varepsilon_1.L$.

Every noisy sensor reading, either missing or not, is represented by one RV in the BN. In addition, conditional probability tables (CPTs) are used to capture probabilistic correlations across sensor readings. Such a generalized model essentially acknowledges the potential for uncertainty in all probabilistic values. Bayesian networks take the traditional SML point of view where models for all random quantities are learned from historical training quantities and further manipulated on the basis of current observations. In addition, note that the joint distribution over the missing-value RVs can be obtained using conventional SML techniques for *conditioning* the BN on the observed *evidence values*, which are the observed sensor readings.

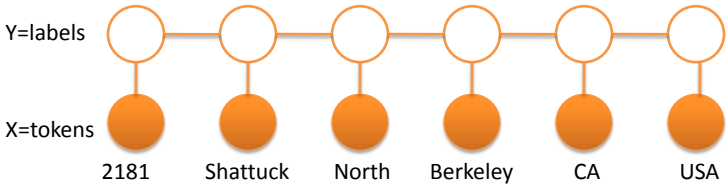
The following example describes a very simple BN between only four sensor readings.

Example 1 *Figure 2.1 depicts a simple* BN*, representing the joint distribution over 4* RV*s* $\mathsf{Pr}[\varepsilon_1.Tp, \varepsilon_1.L, \varepsilon_2.Tp, \varepsilon_2.L]$*, where the 4* RV*s are the temperature (Tp) and light (L) readings of two sensors* ε_1 *and* ε_2*. The local model for each* RV *is given by a conditional probability table (CPT). The figure shows only the CPT for* RV $\varepsilon_1.L$*. If the value of* RV $\varepsilon_2.Tp$ *is known to be* $Cold$ *then it is considered as an observed* RV *from the evidence provided in* Sensor1*. The joint distribution* F *of* Sensor1 *can be computed by calculating the probability* $\mathsf{Pr}[\varepsilon_1.Tp, \varepsilon_1.L, \varepsilon_2.L | \varepsilon_2.Tp = Cold]$ *over the* BN*.* □

2.1.3 Conditional Random Fields

In this section, we introduce a state-of-the-art model—linear-chain conditional random fields (CRF)—for information extraction from unstructured text data. In contrast to Bayesian networks described in the previous section, linear-chain CRFs are undirected (Markov) PGMs.

The linear-chain Conditional Random Field (CRF) [17] is a leading probabilistic model for



(a)

x	2181	Shattuck	North	Berkeley	CA	USA	
y1	**street num**	**street name**	**city**	**city**	**state**	**country**	(0.6)
y2	**street num**	**street name**	**street name**	**city**	**state**	**country**	(0.1)

(b)

Figure 2.2. (a) Example CRF model for an address string; (b) two possible segmentations $\mathbf{y}_1$, $\mathbf{y}_2$.

solving IE tasks. We now describe an example IE task called *field segmentation*, in which a text-string is regarded as a sequence of pertinent fields, and the goal is to tag each token in a text-string with one of the field labels (i.e., field names). Consider the following example.

Example 2 *Fig. 2.2(a) shows a CRF model instantiated over an address string* $\mathbf{x}$ *'2181 Shattuck North Berkeley CA USA'. The possible field labels are* $Y = \{$*apt. num, street num, street name, city, state, country*$\}$. *A segmentation* $\mathbf{y} = \{y_1, ..., y_T\}$ *is one possible way to tag each token in* $\mathbf{x}$ *into one of the field labels in* Y. □

The following definition [18, 17] defines the conditional probabilistic distribution of $\mathbf{y}$ given a specific assignment $\mathbf{x}$ by the CRF model.

Definition 2.1.1 *Let* $\mathbf{X}$, $\mathbf{Y}$ *be random variables,* $\Lambda = \{\lambda_k\} \in R^K$ *be a parameter vector, and* $\{f_k(y_t, y_{t-1}, x_t)\}_{k=1}^K$ *be a set of real-valued feature functions. Then a linear-chain conditional random field is a distribution* $p(\mathbf{y} \mid \mathbf{x})$ *that takes the form:*

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} exp\{\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\}, \tag{2.1}$$

where $Z(\mathbf{x})$ *is a normalization function.* □

Two possible feature functions for the CRF model in Example 2 and in Figure 2.2 are:

$$
\begin{aligned}
f_1(y_t, y_{t-1}, x_t) &= [x_t \textit{ appears in a city list}] \cdot [y_t = \textit{ city}] \\
f_2(y_t, y_{t-1}, x_t) &= [x_t \textit{ is an integer}] \cdot [y_t = \textit{ apt. num}] \\
&\quad \cdot [y_{t-1} = \textit{ street name}]
\end{aligned}
$$

A CRF model represents the probability distribution over all possible segmentations of a text-string d. Fig. 2.2(b) shows two possible segmentations of the text-string d and their probabilities.

2.2 Inference Algorithms

Having described the basic Bayesian network and conditional random fields model, in this section, we now describe four different inference algorithms over PGMs: Viterbi, sum-product, Markov chain Monte Carlo (MCMC) Metropolis Hastings, and MCMC Gibbs sampling. Some of these inference algorithms (e.g., Viterbi) are specialized for linear-chain CRF models; while others are general inference algorithms (e.g., sum-product, MCMC) that can be applied to any PGM model, including both Bayesian networks and linear-chain CRFs.

2.2.1 Top-k Inference on CRF Models

The top-k inference is the most frequently used inference operation over the CRF model. Top-k inference determines the *segmentations* with the top-k highest probabilities given a token sequence $\mathbf{x}$ from a text-string d.

The *Viterbi* dynamic programming algorithm [19] is a key implementation technique for top-k inference in IE applications. For simplicity, we provide the equations to compute the top-1 segmentation. These can be easily extended to compute the top-k. The dynamic programming algorithm computes a two dimensional V matrix, where each cell $V(i, y)$ stores the top-1 partial segmentations ending at position i with label y.

$$
V(i, y) = \begin{cases} \max_{y'}(V(i-1, y') \\ \quad + \sum_{k=1}^{K} \lambda_k f_k(y, y', x_i)), & \text{if} \quad i \geqslant 0 \\ 0, & \text{if} \quad i = -1. \end{cases} \tag{2.2}
$$

We can backtrack through the V matrix to recover the top-1 segmentation sequence $y*$. The complexity of the Viterbi algorithm is $O(T \cdot |Y|^2)$ where T is the length of the text-string and $|Y|$ is the number of labels.

Constrained Top-k Inference: Constrained top-k inference [20] is a special case of top-k inference. It is used when a subset of the token labels has been provided (e.g., via a user interface). Let

$\mathbf{s}$ be the evidence vector $\{s_1, ..., s_T\}$, where s_i is either $NULL$ (i.e., no evidence) or the evidence label for y_i. Constrained top-k inference can be computed by a variant of the Viterbi algorithm which restricts the chosen labels $\mathbf{y}$ to conform with the evidence $\mathbf{s}$.

Marginal Inference: Marginal inference computes a marginal probability $p(y_t, y_{t+1}, ..., y_{t+k}|\mathbf{x})$ over a single label or a sub-sequence of labels [17]. The backward-forward algorithm, a variation of the Viterbi algorithm, is used for such marginal inference tasks.

2.2.2 Sum-Product Algorithm

Sum-product (i.e., belief propagation) is a message passing algorithm for performing inference on graphical models, such as CRF and Bayesian networks [11, 10]. The simplest form of the algorithm is for tree-shaped models, in which case the algorithm computes exact marginal distributions.

The algorithm works by passing real-valued functions called messages along the edges between the nodes. These contain the "influence" that one variable exerts on another. A message from a variable node y_v to its "parent" variable node y_u in a tree-shaped model is computed by summing the product of the messages from all the "child" variables of y_v in $C(y_v)$ and the feature function $f(y_u, y_v)$ between y_v and y_u over variable y_v:

$$\mu_{y_v \to y_u}(y_u) = \sum_{y_v} f(y_u, y_v) \prod_{y_u^* \in C(y_v)} \mu_{y_u^* \to y_v}(y_v). \tag{2.3}$$

Before starting, the algorithm first designates one node as the root; any non-root node which is connected to only one other node is called a leaf. In the first step, messages are passed inwards: starting at the leaves, each node passes a message along the edge towards the root node. This continues until the root has obtained messages from all of its adjoining nodes. The marginal of the root node can be computed at the end of the first step.

The second step involves passing the messages back out to compute the marginal of all the nodes in a PGM: starting at the root, messages computed according to formula 2.3 are passed in the reverse direction, until all leaves have received their messages. Like Viterbi, the complexity of the sum-product algorithm is also $O(T \cdot |Y|^2)$.

Variants of the sum-product algorithm for cyclic models require either an intractable junction-tree step, or a variational approximation such as loopy belief propagation (BP) [11, 21]. In this thesis, we do not study these variants further.

2.2.3 MCMC Inference Algorithms

Markov chain Monte Carlo (MCMC) methods are a class of randomized algorithms for estimating intractable probability distributions over large state spaces by constructing a Markov chain sampling process that converges to the desired distribution. Relative to other sampling methods, the main benefits of MCMC methods are that they (1) replace a difficult sampling procedure from a high-dimensional target distribution $\pi(w)$ that we wish to sample with an easy sampling proce-

```
GIBBS (N)
1  w_0 ← INIT(); w ← w_0; // initialize
2  for idx = 1, ..., N do
3     i ← idx%n; // propose variable to sample next
4     w' ~ π(w_i | w_{-i}) // generate sample
5     return next w' // return a new sample
6     w ← w'; // update current world
7  endfor
```

Figure 2.3. Pseudo-code for MCMC Gibbs sampling algorithm over a model with n variables.

dure from a low-dimensional local distribution $q(\cdot|w)$, and (2) sidestep the $\#P$-hard computational problem of computing a normalization factor. We call $q(\cdot|w)$ a "proposal distribution", which—conditioned on a previous state w—probabilistically produces a new world w' with probability $q(w'|w)$. In essence, we use the proposal distribution to control a random walk among points in the target distribution. We review two MCMC methods here: Gibbs sampling and Metropolis-Hastings (MCMC-MH).

Gibbs Sampling

Let $w = (w_1, w_2, ..., w_n)$ be a set of n random variables, distributed according to $\pi(w)$. The proposal distribution of a specific variable w_i is its marginal distribution $q(\cdot|w) = \pi(w_i|w_{-i})$ conditioned on w_{-i}, which are the current values of the rest of the variables.

The Gibbs sampling algorithm (i.e., Gibbs sampler) first generates the initial world w_0, for example, randomly. Next, samples are drawn for each variable $w_i \in w$ in turn, from the distribution $\pi(w_i|w_{-i})$. Figure 2.3 shows the pseudo-code for the Gibbs sampler that returns N samples. In Line 4, $\sim$ means a new sample w' is drawn according to the proposal distribution $\pi(w_i|w_{-i})$.

Metropolis-Hastings (MCMC-MH)

Like Gibbs, the MCMC-MH algorithm first generates an initial world w_0 (e.g., randomly). Next, samples are drawn from the proposal distribution $w' \sim q(w_i|w)$, where a variable w_i is randomly picked from all variables, and $q(w_i|w)$ is a uniform distribution over all possible values. Different proposal distributions $q(w_i|w)$ can be used, which result in different convergence rates. Lastly,

each resulting sample is either accepted or rejected according to a Bernoulli distribution given by parameter α:

$$\alpha(w', w) = \min(1, \frac{\pi(w')q(w|w')}{\pi(w)q(w'|w)}) \tag{2.4}$$

The acceptance probability is determined by the product of two ratios: the model probability ratio $\pi(w')/\pi(w)$, which captures the relative likelihood of the two worlds; and the proposal distribution ratio $q(w|w')/q(w'|w)$, which eliminates the bias introduced by the proposal distribution.

2.3 Integrating SML with Databases

In the previous two sections, we gave a primer on the two types of PGMs used in this thesis and the corresponding inference algorithms. In the next three sections, we layout three bodies of work that all tried to integrate statistics and probabilities with relational databases and data model.

Research that integrates statistical methods with relational database systems can be traced back to 1990's when the term *data mining* was introduced. Data mining, a relatively young and interdisciplinary field of computer science, is the process of discovering new patterns from large datasets involving methods at the intersection of statistics, artificial intelligence, machine learning, and database systems. Examples include efficient clustering, classification, and association rule mining algorithms and scalable in-database implementations [22, 23, 24]. The more recent research integrating probabilities or SML with databases include work from probabilistic databases, in-database SML methods, and statistical relational learning.

The most relevant work to this thesis is from the probabilistic database (PDB) literature. The PDB research includes: (1) proposals of new data models in order to store data with uncertainties and probabilities; (2) definitions of new query semantics over uncertain data and probability distributions; (3) development of new algorithms for probabilistic relational query processing and optimization; and (4) implementation of such PDB systems. In Section 2.4, we survey key concepts and main works in the PDB literature.

In recent years, more sophisticated models such as PGMs are developed in the SML literature for extracting information, making decisions, and for predicting future events. In Section 2.5, we describe works that integrate PGMs methods with relational database systems. This body of work is the closest to the data mining literature, where the statistical methods are implemented on top of the relational database systems. Different from the PDB work, the relational data model, the relational query semantics, and the relational data processing remain the same for the in-database SML work.

Statistical relational learning (SRL), different from the previous two bodies of work from the database community, is a research direction in the SML community. SRL research develops higher-level and more declarative probabilistic SML models. Moreover, learning algorithms in SRL use relational data from database systems and can take advantage of the structural information from

such data. The Bayesian network variant in Section 3.3.2 for sensor networks is an example SRL model. In Section 2.6, we describe a number of important works in the SRL literature.

2.4 Probabilistic Database Systems

In this section, we first summarize the key concepts, introduced in the past literature, that most or all of the proposed probabilistic database systems share in common. Then we describe in detail a subset of proposed PDBs, their data models, query evaluation methods, and specific contributions. Last, we compare those systems and point out how the approach we take is different from approaches in previous work.

2.4.1 Key Concepts

Probabilistic databases (PDBs) store, manage, and query uncertain data and their probabilities. In this section, we describe some of the key concepts in the probabilistic database literature, including (1) different types of uncertainties; (2) different probabilistic data models; (3) possible world query semantics; and (4) various probabilistic query evaluation approaches.

Two Types of Uncertainties

A probabilistic database represents a set of *possible worlds* (i.e., possible database instances) $W = \{W_1, W_2, ...\}$ and a probability distribution F over them. Two types of uncertainty exist in the data: tuple-level uncertainty and attribute-level uncertainty. Tuple-level uncertainty represents that the existence of a tuple in a data table is uncertain. Attribute-level uncertainty represents that the value of an attribute in a data table is uncertain. Most PDBs only handle tuple-level uncertainty. Although attribute-level uncertainty can be converted into tuple-level uncertainty, the resulting table would contain far more tuples and a great deal of redundant (i.e., un-normalized) information. Thus, supporting attribute-level uncertainty is highly desirable, not only for compact, natural and flexible representation but also for efficient query evaluation.

Three Types of Probabilistic Database (PDB) Models

There are three types of probabilistic database models in the literature to date, which are classified by the type of correlations they allow among the tuples. The first type is the *tuple-independent* probabilistic database, in which the tuples are considered to be independent probabilistic events. A tuple-independent PDB can be designed to handle both tuple-level and attribute-level uncertainties. An example tuple-independent PDB is Trio [9].

The second type is *block independent-disjoint* (BID) probabilistic databases, in which the tuples are partitioned into blocks, such that all tuples within a block are mutually exclusive events, and all tuples from different blocks are independent events. Although mutually exclusive correlation among tuples is allowed, the BID model still cannot express general correlations such as positive correlation among tuples. In many ways, BID is designed to express attribute-level un-

certainty using tuple-level uncertainty and mutually exclusive correlation [25]. An example BID probabilistic database is MystiQ [26, 27].

The third type is the *graphical model* probabilistic database, in which a probabilistic graphical model is natively used in the database to encode correlation among both the tuple-level and the attribute-level uncertainties. Compared to the BID and the tuple-independent PDBs, a probabilistic graphical model can represent much more complex probabilistic correlations that exist among tuples and attribute values. Moreover, it can compactly represent general probability distributions using conditional independence and factorizations.

Although, in theory, both the BID and the tuple-independent PDBs can represent arbitrary probability distributions by storing all possible worlds with their corresponding probabilities, in practice it is infeasible because the number of possible worlds is exponential in the number of uncertain values in the database [28]. In contrast, probability distributions can often be encoded by a simple probabilistic graphical model. Therefore, we advocate graphical model PDBs because they enable compact representation of probability distributions and efficient query processing algorithms. Moreover, all tuple-independent and BID PDBs can be converted into graphical model PDBs [29].

Finally, we can also classify probabilistic models into *discrete* and *continuous*. In the former, attributes are discrete random variables; in the latter, they are continuous random variables. Because it is possible to create PGMs (e.g., Bayesian networks) with continuous random variables, it is possible to extend our work to continuous domains. However, for simplicity, we focus on PGMs over discrete domains in this thesis.

Possible World Query Semantics

To our knowledge, all recent work in query processing for probabilistic databases follows the possible world query semantics. The possible world semantics define the correct answer to any query, which can involve both relational and inference operators, over any PDB system that defines a set of possible worlds W and a probability distribution over them F. The possible world semantics state the following.

Starting from a PDB $\mathcal{DB}^p$, representing a set of possible worlds W and a probability distribution F over them, the resulting $\mathcal{DB}^p$' of a query Q over this PDB $Q(\mathcal{DB}^p\ )$ should represent a new set of possible worlds W' and a new distribution F'. The resulting worlds in W' are the distinct possible worlds generated from applying the query to each of the possible worlds in W. The resulting distribution F' captures the probabilities aggregated over W': the probability of a world $W_n \in W$' is the sum of the probabilities of all the worlds in W that result in W_n after the query.

In tuple-independent PDBs, the output probabilistic database $\mathcal{DB}^p$'(W',F') is represented as a set of pairs (t, p), where t is a possible tuple, i.e., it is in one of the possible resulting worlds W', and p is the probability that t is in W_n when W_n is chosen randomly from the set of possible worlds W'. In other words, p represents the marginal probability of the event the query returns the answer t over the space of possible resulting worlds. This representation of the results does not report how distinct tuples are correlated. Thus, we know that $t1$ is an answer with probability $p1$

and that $t2$ is an answer with probability $p2$, but we do not know the probability that both $t1$ and $t2$ are answers.

However, it has been shown in [30] that starting out with a tuple-independent PDB $\mathcal{DB}^p$, certain relational queries (e.g., aggregation, join, duplicate elimination) can generate a resulting PDB with correlated tuples. Thus, neither the tuple-independent nor the BID model is closed for relational queries under the possible world semantics.

Graphical models are not only a more expressive and compact representation of probabilistic distributions, but they also enable PDBs to be closed under relational and inference queries. Given the possible world semantics, graphical model PDBs can preserve the full joint distribution of the resulting PDB $\mathcal{DB}^p$ without information loss, including the correlations generated during query processing.

Query Evaluation Approaches

Following the possible world query semantics, there are two query evaluation approaches for graphical model PDBs [8]: the extensional query plan and the intensional query plan.

An *extensional query plan* is a query plan that explicitly manipulates probabilities and computes both the answers and probabilities. Two popular examples of extended operators in extensional query plans are the independent join operator, which multiplies the probabilities of the tuples it joins, under the assumption that they are independent, and the independent project operator, which computes the probability of an output tuple, again assuming that the tuples are independent [8]. In general, an extensional plan does not correctly compute the query probabilities. If the plan does correctly compute the output probabilities for every input database, then it is called a *safe query plan*. Safe plans are easily added to a relational database engine, and the complexity of a safe plan is polynomial to the size of the data.

On the other hand, not all queries can be evaluated using the extensional approach and safe query plans. *Intensional query evaluation* first computes the lineage of the query [9], then computes the probability of the lineage expression. Lineage connects uncertainties to other uncertainties in the database, from which they were derived through query processing. The intensional approach reduces query evaluation to the problem of computing the probability of a propositional formula. Intensional evaluation works for every query, but it can perform much worse than extensional query evaluation.

In fact, it has been shown in [29] that both extensional and intensional query evaluation can be mapped to inference operations over probabilistic graphical models representing the probability distribution of the PDB. Safe plans give rise to inference problems with tree-structured graphical models in which running inference is easy. This result coincides with the extensional query processing procedure and complexity. The queries that require intensional query evaluation are those that result in graphical models that contain cycles. Inference over cyclic graphical models is known to be NP-hard, which coincides with the findings for intensional query evaluation.

2.4.2 PDB Research Projects

Probabilistic databases and new data models to capture uncertain data were active area of research in the mid 1980's and the 1990's [31, 32, 33, 34, 35]. Data models with attribute-level uncertainties [33], semantics similar to possible world semantics [34], and query processing techniques with a relaxed version of possible world semantics [35] were all proposed. During this period, the publications were somewhat sporadic with little effort put into synthesizing these works, each with different data models and query semantics. Moreover, there was no published effort to build a prototype probabilistic database system for a realistic application.

More recently (since the year 2000), with the demand from many applications to handle data with uncertainty and to apply probabilistic data analysis methods to the data, an increasing need for a data management system that can deal with uncertain data, apply probabilistic methods, and query over probability distributions has been recognized. A number of probabilistic data representations and systems have been proposed [26, 36, 9, 37, 29, 38, 39, 40].

In contrast to the earlier work, the more recent work in probabilistic databases is more application-driven and system oriented. A number of PDB systems have been proposed and prototyped during the past seven years. These different projects on PDB enabled synthesis and comparison between different approaches and this competition helped the field of probabilistic databases to develop tremendously within a very short time. In this section, we survey the more recent work in the probabilistic database literature that is closely related to the work reported in this thesis.

MystiQ: Suciu et al. in [26] and in a series of papers that followed studied the problem of efficiently answering queries on probabilistic databases, returning a list of tuples ranked by their probability of existence in the database. This line of work gave birth to MystiQ [8], a system that uses probabilistic query semantics to find answers in data sources that contain uncertainties.

On one hand, being the first work in the recent probabilistic databases literature, this work focused on a very limited scope for probabilistic data representation and query processing, supporting tuple-level uncertainty, tuple-independence, and extensional query evaluation. These assumptions make this work hard to apply to real applications [28]. On the other hand, this work established a solid foundation for follow-up work by defining correct query semantics and laying out a dichotomy of query evaluation methods. The main contributions of this work include the following.

First, this work revisited the semantics of traditional relational query operators in order to enable them to handle values with probabilities, formalized the notion of possible worlds as all the possible instantiations of a probabilistic database, and formalized the standard possible worlds query semantics described in Section 2.4.1. These semantics have been used by most of the later work on query processing in PDB.

Second, as demonstrated by Dalvi and Suciu [26], probabilistic query processing quickly gives rise to *computationally-intractable* probabilistic inference problems, as complex correlation patterns can emerge during processing even if naive independence assumptions are made on the base data tuples. In fact, modulo a class of "safe" query execution plans, query processing in a tuple-

independent PDB is $\#P$-complete in the size of the database (i.e., number of tuples) [1]. Finally, approximate algorithms are provided in [26] for queries that do not have a safe query plan.

Trio: The Trio [9] project focused on promoting *data lineage* as a first-class citizen in an extension of the relational model and an SQL-based query language. The probabilistic data representation and the probabilistic query processing was formalized using data lineage.

Similar to MystiQ, *X-relations*, the probabilistic representation in Trio, assumed tuple-level independence. In an X-relation, the uncertainties can exist both in tuple existence and in attribute values. Unlike MystiQ, Trio adopted an intensional approach for query processing, in which the probabilities are computed from the Boolean expressions generated from lineage. In Trio, probabilistic correlations between tuples can be generated from queries and composition of basic events. However, the tuple-independent representation based on X-relations is expensive and unnatural to represent correlations: a large number of base events and a complex Boolean expression may be needed to encode a simple correlation between two random variables.

Orion: A different approach from both MystiQ and Trio is taken in ORION [36, 41], a PostgreSQL-based database system that handles uncertainty and probabilistic queries for constantly evolving data. Its authors are motivated by the uncertainty that characterizes today's sensor network deployments.

They propose a probabilistic data model that represents each data value as an interval and a (continuous) probability distribution, and they extend the semantics of common SQL operators to support the execution of probabilistic queries conforming to this proposed model. The system is designed to handle uncertainties in both attribute values and tuple existence. Orion 2.0 [41] can support correlations between attribute values within a single tuple. However, it cannot support correlations between tuples. Thus, Orion can only support tuple-independent uncertainties in the data.

The Orion system uses both extensional and intensional query evaluation. For queries with safe plans, query evaluations similar to extensional query plans are performed; for queries without a safe plan, query evaluations similar to intensional query plans are used. In [42], the Orion system is extended to allow a form of probabilistic joins, namely *probabilistic threshold joins* between probabilistic relations.

PrDB: More recent work has attempted to address the limitations imposed by the oversimplifying of independence assumptions between tuple existence and attribute values by explicitly modeling the correlations among the entities stored in the database. Sen et al. [29] made use of probabilistic graphical models from the statistical learning literature to represent and maintain the correlations between tuple-level uncertainties in the base tables and during query processing. Thus, the resulting graphical models contain one random variable per tuple in both the base tables and the generated tables from relational queries. PrDB extended the probabilistic data model in [43] to also support attribute value uncertainty.

In PrDB, query processing over a PDB based on graphical models is reduced to manipulation of the probabilistic graphical models and inference over the resulting models. This work also

[1] This problem, as shown by Sen and Deshpande [29], is isomorphic to well-known intractability issues faced in traditional SML work.

described the in-database implementation of a junction tree inference algorithm, and another first-order inference algorithm over probabilistic graphical models [30].

This work came out about the same time as our work, and used a similar approach as ours in probabilistic data modeling and query processing. However, we have developed an integrated PDA system with inference and relational query processing as well as optimization. In contrast, the PrDB project did not explore how to integrate the individual techniques being considered (e.g., inference algorithms, index structures) in a unified framework or implementation. Also, the techniques developed in PrDB could not support some specific PDA applications, in particular IE. This is because the IE-models are not general graphical models, but rather are linear-chain models, which require special inference and index techniques.

MCDB: MCDB [40] is a probabilistic database built on Monte Carlo algorithms. The driving application of MCDB is risk management: statistical and probabilistic analysis using what-if queries to calculate cost and benefit under different conditions. This application is very important for financial institutions, such as banks and insurance companies.

MCDB provides a rich data model by combining two simple primitives: a large, predefined set of random variables and SQL queries (views). An MCDB database does not encode uncertainty in the data itself. Instead, it allows the user to define arbitrary variable generation functions, VG, which are pseudorandom generators for any random variable. The semantics of MCDB is the standard possible worlds semantics for probabilistic databases. The SQL queries can define views that join a deterministic table and the probabilistic values generated from a VG function. An example VG function is Normal(mean,stdv), which is a variable generating function that generates a normal distribution with a given mean and standard deviation.

For query evaluation, MCDBs run Monte Carlo simulations. This has the major advantage that it is a uniform approach applicable to all, even very general, probabilistic models. The disadvantage is that Monte Carlo simulations are costly and cannot handle the tail of the distribution effectively.

Recent work from this project, also looks at the scalability problem of such a sampling based system [44] and the techniques to deal with questions asking for tail distributions [45]. However, scalability and tail distributions remain to be the two major concerns for any system built on Monte Carlo methods.

MayBMS: MayBMS [39] can support both tuple-level and attribute-level uncertainty using U-relations. U-relations represent uncertainty and correlations between attribute values by decomposing a large joint distribution into small factor tables. This factorization results in a more succinct representation of the probability distribution than the world-set decompositions (WSDs), the probabilistic data representation in [46], and the uncertainty-lineage databases (ULDBs) such as Trio. Such factorizations are similar to those used to generate a probabilistic graphical model from a high-dimensional distribution based on conditional independence.

Like Trio, the MayBMS system also supports intensional query evaluation. But using U-relations, MayBMS results in more efficient query processing. U-relations allow for a large class of queries to be processed using relational algebra only, and thus efficiently in the size of the data.

MayBMS is a significant step forward from earlier probabilistic databases. However, U-

relations do not directly support SML model representations. There is a gap between the U-relation data model and the SML-based probabilistic data analysis model. For example, the linear-chain CRF model for IE cannot be directly represented using U-relations. Moreover, it is unclear how to efficiently and compactly transform the probabilistic distribution encoded by SML models over uncertain data to the U-relations supported by MayBMS. Finally, statistical inference algorithms are not supported in MayBMS.

2.4.3 Summary

Recent research efforts in PDB have injected new excitement into the area of uncertainty management in database systems. When we started this thesis work, most PDB work relied on somewhat simplistic models of uncertainty and placed the focus on simple probabilistic extensions that can be easily mapped to existing relational database architectures, essentially ignoring the state-of-the-art in SML. Given the background of prior work, the goal of our work is to build, on the foundation of possible world semantics and probabilistic graphical models, a probabilistic database that (1) supports graphical models natively to represent the uncertainties and probability distributions in data; (2) implements statistical inference operations in addition to the relational operations; (3) supports probabilistic relational operators based on the possible world semantics; and (4) achieves a deep integration and query optimization between the inference and relational operations; so that it can efficiently and accurately support various PDA tasks based on SML models.

2.5 In-database SML Methods

The need for statistical methods inside a relational database system was acknowledged by the development of data mining techniques to support processing of large datasets that involved operations outside of the realm of standard transactional query processing such as classification, clustering, and association rule mining [47, 48].

In this section, we describe research that has considered integrating SML models, such as PGMs, into relational database systems. In contrast to the PDB work in the previous section, this body of work does not study how to represent uncertain data and probability distribution in the database or how to perform query processing over uncertainties and probabilities. Instead, statistical models and methods are implemented in the database as UDFs to perform certain analysis over data, where the main challenge is how to implement and apply such statistical models effectively over the data in database systems.

BBQ: Deshpande et al. [5] proposed a robust query processing architecture called BBQ for sensor network applications. BBQ uses multivariate Gaussian distributions to handle imprecisions in sensor readings and to perform approximate query answering. BBQ was one of the most influential papers that attempted to marry the worlds of probabilistic inference and query processing. However, BBQ models the correlation of a large number of random variables through a single multivariate distribution, which leads to exponentially large probability representation and inefficient probabilistic reasoning.

MauveDB: MAUVEDB [37] is the first known database management system that explores the

idea of building statistical models of the stored data, in order to avoid problems of data irregularities, noise, or missing values by maintaining a more robust representation of the "physical phenomenon" from which the original data were sampled. This work also discusses issues related to the creation and maintenance of regression and interpolation models as "model-based views" in a user-transparent way, which can be later accessed using traditional SQL queries. The semantics of the system could support any sort of probabilistic model in the future, by defining a model-based view appropriately, along with the necessary query language extensions and the UDFs that will handle the statistical parts of the computation.

MPF Queries: Bravo and Ramakrishnan [49] have studied MPF (marginalize a product function) queries, which are aggregate queries over a stylized join of several relations. In probabilistic inference, this join corresponds to taking the product of several probability distributions, while the aggregate operation corresponds to marginalization. Probabilistic inference (e.g., sum-product) algorithms can be expressed directly as MPF queries in a relational setting. Therefore, optimizing the evaluation of MPF queries provides scalable support for probabilistic inference in database systems. They built on ideas from database query optimization, as well as traditional algorithms such as variable elimination and belief propagation from the probabilistic inference literature.

Tuffy: A number of techniques are implemented in Tuffy [50] to scale up inference algorithms over Markov Logic Networks (MLNs) [51, 52, 53, 54]. The techniques include: (1) a solution that pushes MLNs into RDBMSes using bottom-up grounding that leverages the RDBMS optimizer; (2) a hybrid architecture to support both disk-based grounding, as well as in-memory inference; and (3) data partition techniques to decrease the memory usage and increase parallelism and scalability of the in-memory inference algorithms. Results show orders-of-magnitude improvements in runtime compared to Alchemy [55], a software package that provides a series of algorithms for statistical relational learning and probabilistic logic inference, based on the Markov logic network representation.

MLNs and Tuffy are geared towards a variety of text analysis applications, including information extraction, entity resolution, and text mining. Apart from PrDB and our work, Tuffy is a new attempt to leverage the efficient data processing power in databases to scale up the SML inference algorithms. It also took the approach to natively support in-database inference algorithms to take advantage of the database components, such as indexes and the query optimizer.

Tuffy, however, is not a probabilistic database: it does not aim to support relational queries over the probabilistic output of SML inference. Rather, it aims to be a scalable MLN inference engine utilizing database facilities. Therefore, Tuffy does not support or optimize probabilistic data analysis queries that involve both relational and inference operations over SML models.

2.6 Statistical Relational Learning

As exemplified by recent research on *statistical relational learning*, the SML community has also been moving in the direction of higher-level, more declarative probabilistic modeling tools. The key idea lies in learning and reasoning with *first-order (FO)* (or, *relational*) probabilistic models in which the random variables in the model representation correspond to *sets of random parameters* in the underlying data [56, 57, 58].

PRMs: The *Probabilistic Relational Model (PRM)* [56, 59] is a good example of such a model. PRMs are an FO extension of traditional (propositional) BNs over a relational DB schema: random variables in the PRM directly correspond to attributes in the schema; thus, PRMs can capture correlation structure across attributes of the same or different relations (through foreign-key paths) *at the schema level*. The idea, of course, is that this correlation structure is *shared* (or, FO-quantified) across all data tuples in the relation(s). For instance, in a `Person` database, a PRM can specify a correlation pattern between `Person.Weight` and `Person.Height` (e.g., a conditional probability distribution $\mathsf{Pr}[\texttt{Person.Weight} = x|\texttt{Person.Height} = y]$ that is shared across all `Person` instances. Such shared correlation structures not only allow for *more concise and intuitive* statistical models, but also enable the use of more efficient *lifted probabilistic inference* techniques [58] that work directly off the concise FO model.

Various extensions of the PRM formalism have been proposed to express simple correlations between objects of different relations, to capture the existence uncertainty of an object (tuple), or to express even richer formalisms that support *class hierarchies* [60].

MLNs: A Markov logic network (MLN) [61] is a probabilistic logic that applies the ideas of a Markov network to first-order logic, enabling uncertain inference. An MLN model is a collection of formulas from first-order logic, to each of which is assigned a real number, the weight. Taken as a Markov network, the vertices of the network graph are atomic formulas, and the edges are the logical connectives used to construct the formula. A potential function is associated with each formula and takes the value of 1 when the formula is true, and 0 when it is false. The potential function is combined with the weight to form the Gibbs measure and partition function for the Markov network.

The above definition glosses over a subtle point: atomic formulas do not have truth values unless they are grounded and given an interpretation—that is, until they are ground atoms with a Herbrand interpretation. Thus, a Markov logic network becomes a Markov network only with respect to a specific grounding and interpretation; the resulting Markov network is called the ground Markov network. The vertices of the graph of the ground Markov network are the ground atoms. The size of the resulting Markov network thus depends strongly (exponentially) on the number of constants in the domain of discourse.

Inference in MLNs can be performed using standard Markov network inference techniques over the minimal subset of the relevant Markov network required for answering the query. These techniques include Gibbs sampling and belief propagation.

Dyna: Dyna [62, 63] is a small, very-high-level programming language that makes it easy to specify dynamic programs and train their weights. Given a short declarative specification in Dyna, the Dyna optimizing compiler produces efficient C++ classes that form the core of the C++ application.

Natural applications of Dyna include various kinds of parsing, machine translation, speech decoding, and finite-state modeling. Dyna also has many other applications, especially in other applied AI areas, as dynamic programming (and its degenerate case, tree search) is a common strategy for solving combinatorial optimization problems.

Properly speaking, Dyna is a "weighted" logic programming language: terms have values, and Horn clauses are replaced by aggregation equations. It is Turing-complete. Dyna can also be thought of as extending compiled C++ with powerful deductive database features.

2.7 Summary

To summarize, previous work in probabilistic database literature has laid a solid foundation for probabilistic database research in terms of uncertainty representation, query semantics and query evaluation methods. However, most work before this thesis placed the focus on simple probabilistic extensions that can be easily mapped to existing relational database architectures and essentially ignored the state-of-the-art research in statistical machine learning literature. Other work in the database literature has been done to incorporate advanced statistical models and methods into the database systems. Moreover, the work on statistical relational learning in the SML literature develops models and algorithms that can take advantage of first-order structures in relational data. However, none of these two bodies of work studies the problem of storage and query processing over the uncertain data and the probabilistic outcome of such statistical analysis. The goal of our work is to build upon these prior efforts to natively support statistical machine learning models and algorithms in the database.

Chapter 3

The BAYESSTORE Data Model

In this chapter, we first provide an overview of BAYESSTORE and its system architecture. Then, we outline the BAYESSTORE data model and give two instances of the data model: a first-order Bayesian network (FOBN) model for sensor network applications and a linear-chain conditional random fields (CRF) model for information extraction applications over unstructured text.

The FOBN model, like Bayesian networks (BN), is a general probabilistic graphical model. Moreover, FOBN can capture common correlations in Bayesian networks using first-order templates to improve the compactness of the model. The first-order templates are additional complex components in the model. On the other hand, the linear-chain CRF model has a simple and very restricted probabilistic correlation structure (i.e., linear-chain). Thus, the representation of the CRF model is much simpler than that of the FOBN model. These two BAYESSTORE instances provide a spectrum—general and complex to specific and simple—of probabilistic graphical models that can be represented in BAYESSTORE to perform probabilistic data analysis for two different applications.

3.1 BAYESSTORE Overview

The BAYESSTORE probabilistic database system supports uncertain data and probabilistic graphical models as first-class citizens. The uncertain data is stored in incomplete relations as missing values, and the graphical models are stored in separate relational data tables.

On top of the relational and the probabilistic components, an extended set of operators is supported, including the probabilistic relational operators over uncertain data and inference operators over the probabilistic graphical model. These new operators together with deterministic relational operators are used to compose complex probabilistic analysis queries, which are compiled into query plans and are executed by the BAYESSTORE PDB. The results of the analysis queries, when appropriate, are associated with uncertainties and probabilities quantified over them.

Figure 3.1 shows the architecture of BAYESSTORE. The data, either structured or unstructured, can be stored directly in BAYESSTORE in a relational format. BAYESSTORE supports both

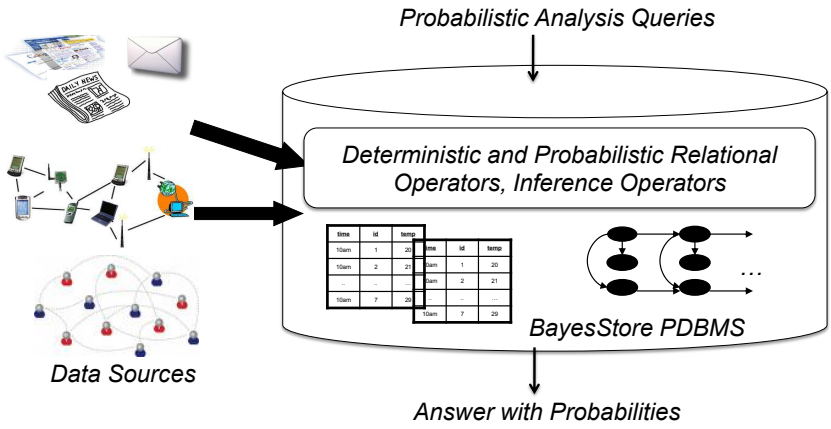


Figure 3.1. The system architecture for BAYESSTORE Probabilistic Database System.

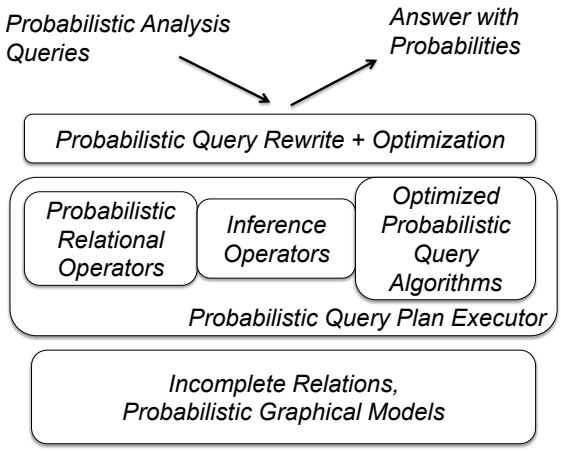


Figure 3.2. The block diagram of the BAYESSTORE PDB components built on top of the PostgreSQL DBMS.

relational and inference operations. Using BAYESSTORE, probabilistic analysis tasks can be composed as queries, involving both statistical inference and relational operations, over both uncertain and deterministic data. Queries over uncertain data can return either the top-k highest-probability answers or the marginal distributions of the uncertain values.

BAYESSTORE is implemented as a modification of the PostgreSQL 8.4 relational database system. BAYESSTORE extends three components in the database system: the data model, the query plan executor, and the query rewriter and optimizer. Figure 3.2 shows the block diagram of the components we implement in BAYESSTORE to natively support in-database probabilistic models, statistical inference and probabilistic query processing. First, we extend the relational model of traditional databases to support incomplete relations and probabilistic graphical models.

Second, we extend the query plan executor to support: (1) existing algorithms that implement statistical inference algorithms over probabilistic graphical models; (2) new algorithms that imple-

ment probabilistic relational operations over the incomplete relation and the probabilistic model; and (3) optimized algorithms that implement probabilistic queries, achieving a deep integration between the relational and the inference operators, to improve runtime performance. Finally, we extend the query rewriter/optimizer with new rules for probabilistic analysis queries, which for example, choose among different statistical inference algorithms and pick the one that is the most efficient and accurate given a specific query and data.

This chapter describes the BAYESSTORE data model. The following chapters are dedicated to each of the system blocks in Figure 3.2.

3.2 BAYESSTORE Data Model: Incomplete Relations and Probability Distribution

BAYESSTORE is founded on a novel data model that treats uncertain relational data and probabilistic models of uncertainty as first-class citizens in the probabilistic database system. Abstractly, a BAYESSTORE probabilistic database $\mathcal{DB}^p = <\mathcal{R}, \mathrm{F}>$ consists of two key components: (1) A collection of *incomplete relations* $\mathcal{R}$, and (2) A *probability distribution function* F that quantifies the uncertainty associated with all incomplete relations in $\mathcal{R}$.

An incomplete relation $R \in \mathcal{R}$ is defined over a schema $\mathcal{A}^d \cup \mathcal{A}^p$ comprising a (non-empty) subset $\mathcal{A}^d$ of *deterministic attributes* (that includes all candidate and foreign key attributes in R), and a subset $\mathcal{A}^p$ of *probabilistic attributes*. Deterministic attributes have no uncertainty associated with any of their values — they always contain a legal value from their domain, or the traditional SQL NULL. On the other hand, the values of probabilistic attributes may be present or *missing* from R. Given a tuple $t \in$R, non-missing values for probabilistic attributes are considered *evidence*, representing our partial knowledge of t. Missing values for probabilistic attribute $A_i \in \mathcal{A}^p$ capture attribute-level uncertainty; formally, each such missing value is associated with an RV X_j ranging over $\mathsf{dom}(A_i)$ (the domain of attribute A_i). (Note that applications requiring *tuple-existence uncertainty* can incorporate a probabilistic boolean attribute $\texttt{Exist}^p$ to capture the uncertainty of each tuple's *existence* in an incomplete relation. As discussed later, this requirement can also arise during query processing in BAYESSTORE.)

The second component of a BAYESSTORE database $\mathcal{DB}^p = <\mathcal{R}, F>$ is a probability distribution function F, that models the *joint distribution* of all missing-value RVs for relations in $\mathcal{R}$. Thus, assuming n such RVs, $\mathrm{X}_1, \ldots, \mathrm{X}_n$, F denotes the joint probability distribution $\mathsf{Pr}(\mathrm{X}_1, \ldots, \mathrm{X}_n)$. The association of F with conventional PDB *possible-worlds semantics* [26] is now straightforward: every complete assignment of values to all X_i's maps to a *single* possible world for $\mathcal{DB}^p$.

Note that, unlike most PDB work to date [32, 34, 26, 9, 39], the BAYESSTORE data model employs a clean separation of the relational and probabilistic components: Rather than attaching probabilities to uncertain database elements, BAYESSTORE maintains the probability distribution F over the incomplete relations as a separate entity. This design has a number of benefits. First, it allows us to expose all probabilistic information as a separate, first-class database object that can be queried and manipulated by users/applications. Second, and perhaps most important, it allows us to leverage ideas from state-of-the-art SML techniques in order to effectively *represent,*

query, and manipulate the joint probability distribution F. Thus, the BAYESSTORE data model directly captures the PDB possible-worlds semantics, while exposing the (potentially complex) probabilistic and correlation structures in the data as as first-class objects that can be effectively modeled and manipulated using SML tools.

From our definition of F, it is evident that it is a potentially huge mathematical object — the straightforward method for representing and storing this n-dimensional distribution F is essentially linear in the number of possible worlds of $\mathcal{DB}^p$.

3.3 BAYESSTORE Instance for Sensor Networks

In this section, we first describe a BAYESSTORE instance, which captures uncertain sensor data using incomplete relations and the probabilistic distribution F over noisy sensor readings using first-order Bayesian networks. A variation of Bayesian networks, FOBN extends BNs with a set of first-order extensions. Such extensions enable FOBN to obtain more compact probabilistic model representations than BNs through specifying shared correlation and conditional independence structures.

3.3.1 Sensor Tables: The Incomplete Relations

Suppose there are two sets of environmental sensors in different rooms of a hotel. The first set of sensors monitors temperature and light levels; while the second monitors humidity and light levels. Sensor readings are inherently noisy – readings may be dropped due to sensor malfunctions or power fluctuations, or even garbled by radio interference from other equipment. For the ease of exposition, we assume that sensor readings are discretized into binary values; $Cold$ and Hot for temperature, Drk (Dark) and Brt (Bright) for light, and $High$ and Low for humidity levels. We regard all collected readings from the two sets of sensors being stored in a database $\mathcal{DB}^p$ of two incomplete relations respectively.

Sensor1(Time(T), Room(R), Sid, temp^p(T_p), light^p(L))

Sensor2(Time(T), Room(R), humidity^p(H), light^p(L))

Probabilistic attributes are denoted with superscript p, while the attributes constituting the primary key are underlined. In this example, neither relation is associated with tuple-level existence uncertainty. Figure 3.3(a) depicts an instance of these relations. Blank cells correspond to missing values, and are represented by an RV for each, taking values from the domain of the corresponding probabilistic attribute. Each RV can be referred to by its tuple identifier and attribute name (e.g. $\varepsilon_1.Tp$ is the RV of the missing value of tuple ε_1).

In this snapshot of $\mathcal{DB}^p$, the possible worlds distribution F can be captured by a 16-dimensional distribution table, over the RVs associated with the 16 values under the probabilistic attributes in *Sensor1* and *Sensor2*. The Bayesian networks, introduced in Section 2.1.2, can be used to more compactly represent this 16-dimensional distribution using conditional indecency and factorizations. For example, the Figure 3.3(b) shows the BN over the first 4 uncertain values in *Sensor1*,

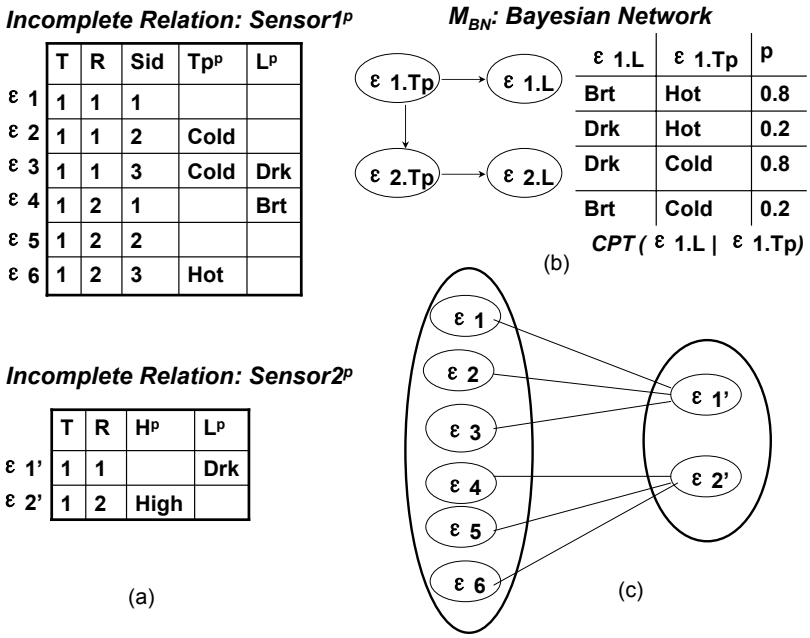


Figure 3.3. (a) Incomplete Relations *Sensor1* and *Sensor2*. Each tuple is referred to by its identifier (ε_i). (b) An example Bayesian Network representation of the probabilistic distribution of tuples $\{\varepsilon_1, \varepsilon_2\}$, and the CPT associated with RV $\varepsilon_1.L$. (c) The mapping between the entity sets of two stripes, S_{T_p} and S_H, from *Sensor1* and *Sensor2* respectively, involved in a child-parent relationship.

where the 4-dimensional distribution is reduced into 3 two-dimensional local distributions. If the rest of the RV's observe similar conditional independence, then the 16-dimensional distribution (with size 2^{16}) can be reduced into 12 two-dimensional local distributions (with size 12×2^2).

3.3.2 First-Order Bayesian Networks: The Probability Distribution

Even with the use of a factored representation that exploits conditional independence among tuple variables, the size of the resulting tuple-based BN model (also referred to as a *propositional* BN) remains problematic: The model requires one RV per probabilistic attribute, per tuple, per relation! The number of RVs in a propositional BN model is linear to the number of tuples in the database. Thus, the size of the BN model is large. Furthermore, the cost of probabilistic reasoning over these models is bound to be prohibitive for non trivially-sized domains [29].

To overcome these problems, BAYESSTORE employs a novel refinement of recent ideas in the area of *First-Order (FO)* probabilistic models [56, 57, 58]. More specifically, the BAYESSTORE probabilistic model is based on a novel class of *First-Order Bayesian Network (FOBN)* models. In general, a FOBN model $\mathcal{M}_{FOBN}$ extends traditional (propositional) BN structures by capturing independence relationships between *sets* of RVs. This allows correlation structures that are *shared* across multiple propositional RVs to be captured in a concise manner. More formally, a $\mathcal{M}_{FOBN}$ encodes a joint probability distribution by a set of *FO factors* $\mathcal{M}_{FOBN} = \{\phi_1, ..., \phi_n\}$ — each such factor ϕ_i represents the local CPT model of a *population* of (propositional) RVs. For instance, the

popular *Probabilistic Relational Models (PRMs)* [56] are an instance of FOBN models specified over a given relational schema: First-order RVs in a PRM directly correspond to schema attributes, and the correlation structure specified is shared across *all propositional instances* in the relation (which are assumed to be independent across tuples — i.e., no vertical correlations). Attributes in different relation schemas can also be correlated along key/foreign-key paths [56].

The BAYESSTORE FOBN model is based on a non-trivial extension to PRMs that allows correlation structure to be specified and shared across first-order RVs corresponding to arbitrary, declaratively specified, sub-populations of tuples in a relational schema. Thus, unlike PRMs, shared correlation structures in BAYESSTORE can be specified in a very flexible manner, without being tied to a specific relational schema design. This flexibility can, in turn, enable concise, intuitive representations of probabilistic information. The key, of course, is to ensure that such flexibly-defined FO correlation structures are also guaranteed to map to a *valid probabilistic model* over the base (propositional) RVs in the BAYESSTORE database. (Such a mapping is known as *FO model "grounding"* in the SML literature.) In what follows, we give a brief overview of some of the key concepts underlying the BAYESSTORE FOBN uncertainty model, and how they ensure a valid mapping to a possible-worlds distribution.

Entities and Entity Sets The following definition provides a declarative means of specifying arbitrary sub-populations of tuples by selecting "slices" of the deterministic part of an incomplete relation R.

Definition 3.3.1 *[Entity Set $R.\mathcal{E}$] An Entity Set ($R.\mathcal{E}$) of an incomplete relation R with deterministic attributes $K_1, \ldots, K_n$, is the deterministic relation $R.\mathcal{E}$ ($\mathcal{K}'$) (where $\mathcal{K}' \subseteq \{K_1, \ldots, K_n\}$), defined as a relational algebra query which performs a selection and a subsequent projection on any part $\mathcal{K}'$ of $\{K_1, \ldots, K_n\}$: $R.\mathcal{E} \stackrel{def}{=} \pi_{\mathcal{K}'}(\sigma_{condition\ on\ \mathcal{K}'}(\ R\ ))$.* □

The elements of an entity set essentially identify a number of "entities" at arbitrary granularities – tuples or tuple *groups* – which contain some probabilistic attributes that we intend to model. Note that, in general, since entity sets are defined as *projections* on parts of R's superkey, a particular entity ε can correspond to a *set of tuples* in R. The probabilistic attribute RV associated with ε corresponds, in fact, to *the same* RV across all R tuples corresponding to ε; in other words, in any possible world, a probabilistic attribute associated with an entity is *instantiated to the same value* for all tuples in the entity's extent. Thus, intuitively, entities define the granularity of the base, propositional RVs in the BAYESSTORE model. An *entity set* then naturally specifies the granularity of *first-order* RVs (quantified over all entities in the set).

Various entity sets can be specified for the same relation, depending on the level in which entities are captured. Among those, the *maximal* entity set of R, which associates an entity *with each tuple* in R, can be obtained by projecting on any key of R— for our *Sensor1* table example: $\textit{Sensor1}.\mathcal{E}_M = \pi_{\{R,T,Sid\}}(\textit{Sensor1}\ )$. Each entity ε in $\textit{Sensor1}.\mathcal{E}_M$ (i.e. each sensor reading tuple) is associated with two uncertain quantities, its temperature ($\varepsilon.Tp$) and light readings ($\varepsilon.L$), which can be represented by the RVs $\varepsilon.X_{Tp}$ and $\varepsilon.X_L$, that model the stochastic process of assigning values to $\varepsilon.Tp$ and $\varepsilon.L$, respectively.

Coarser entity definitions are also useful when we need to associate the same RV for several tuples in an incomplete relation, i.e., tuples that always have the same probabilistic attribute value in any possible-world instantiation. Such situations arise naturally, for instance, when values of a probabilistic attribute are redundantly repeated in a table or during the processing of cross-product and join operations over probabilistic attributes (Section 4.4).

First-Order RVs and Stripes: Having specified the concept of entity sets, we can now naturally define the notion of first-order RVs in BAYESSTORE. Such FO RVs (termed *stripes*) represent the values of a probabilistic attribute for a *population of entities* that share the same probabilistic characteristics.

Definition 3.3.2 *[Stripe $S <R.\mathcal{A}^p, \mathcal{E} >$] A stripe $S <R.\mathcal{A}^p, \mathcal{E} >$ over an entity set $\mathcal{E}$ of an incomplete relation R, represents a set of random variables, one per entity from $\mathcal{E}$, which models the stochastic process of assigning values to the probabilistic attribute $\mathcal{A}^p$ of that entity.*

Mapping and First-Order Factors: As mentioned earlier, a FO factor in a FOBN captures the shared correlation structure (CPT) for a group of underlying propositional RVs (i.e., entities). Using our earlier definitions, we can define BAYESSTORE *FO factor* $\phi\ (S|Pa(S))$, as a local CPT model that describes the conditional dependence relationship between a *child stripe* S and its *parent stripes* $Pa(S)$. For example, if for all entities in of the incomplete relation *Sensor1*, the RV corresponding to light, L_i, conditionally depends on the temperature, Tp_i, of the same entity ε_i (i.e., $Tp_i \rightarrow L_i$), and this correlation pattern (i.e., CPT) is shared across all entities, then this situation can be captured concisely by two stripes: $S_{Tp} =< Sensor.Tp, \{\pi_{R,T,Sid}Sensor\} >$ and $S_L =< Sensor.L, \{\pi_{R,T,Sid}Sensor\} >$, and a FO factor over them: $\phi\ (S_L|S_{Tp})$.

In the above example, we implicitly assumed the existence of a one-to-one mapping between entities in the child stripe (S_L) and entities in its parent stripe (S_{Tp}); i.e., each (propositional) RV $L_i \in S_L$ will have as a parent its corresponding $Tp_i \in S_{Tp}$. Although this is often the case, non-bijective types of stripe mappings can be defined as well.

Definition 3.3.3 *[Mapping between Stripes f [S_p, S_c]] A mapping f between two stripes – a child stripe $S_c < R.\mathcal{A}^p_c, \mathcal{E}_c >$ and a parent stripe $S_p < R.\mathcal{A}^p_p, \mathcal{E}_p >$, is a surjective function from the entity set $\mathcal{E}_c$ to $\mathcal{E}_p$, $f[S_p, S_c]$: $\mathcal{E}_c \rightarrow \mathcal{E}_p$, expressed as a first-order logic formula over the schemata $\mathcal{K}_c$ and $\mathcal{K}_p$ of the entity sets $\mathcal{E}_c$ and $\mathcal{E}_p$, where $\mathcal{K}_p \subseteq \mathcal{K}_c$. For the mapping to be valid, the selection predicates of the relational algebra expressions used to define $\mathcal{E}_c$ and $\mathcal{E}_p$ respectively, need to be the same.* □

Definition 3.3.3 formalizes the association of the individual elements (RVs) of two stripes involved in a "child-parent" relationship ($S_p \rightarrow S_c$) at a per-instance level. In essence, it requires

that every RV from S_c will have a *single* corresponding parent RV from S_p. In addition, it ensures the minimality of the parent stripe S_p, through the constraint of surjectivity, as every RV in S_p has to be associated with *at least* one RV from S_c. From the above, while it is permissible for an RV from S_p to be parent of more than one RV from S_c, a child RV from S_c cannot have more than one parent (since such multi-variable correlations cannot be captured by a single FO edge).

The last requirement of definition 3.3.3 essentially states that between the entity set attributes of the child and the parent stripes, there is a key-foreign key relationship, so that the entities between the two sets can be unambiguously associated with each other. To continue the previous example, the "1-1" mapping between the stripe S_L and its parent S_{Tp}, can be expressed as:

$$\begin{aligned}(\forall \varepsilon_L \in S_L.\mathcal{E}, \forall \varepsilon_{Tp} \in S_{Tp}.\mathcal{E}) Pa(r.\varepsilon_L) = r.\varepsilon_{Tp}, \text{ s.t.:} \\ \varepsilon_L.T = \varepsilon_{Tp}.T \wedge \varepsilon_L.R = \varepsilon_{Tp}.R \wedge \varepsilon_L.Sid = \varepsilon_{Tp}.Sid\end{aligned} \tag{3.1}$$

where $r.\varepsilon_L$ signifies the RV associated with the entity ε_L from stripe S_L's entity set, and $Pa(r.\varepsilon_L)$ the parent RV of $r.\varepsilon_L$.

Up to this point we have tacitly assumed that we can express correlations only among stripes of the same incomplete relation. Nevertheless, conditional dependence relationships *can* be formed between stripes of different relations. As noted earlier, the two relations must be involved in a key-foreign key relationship, the foreign key of the parent incomplete relation should be part of the primary key of the child, and the selection predicates (if any), populating the entity sets of the stripes must be the same, for the last condition of Definition 3.3.3 to be satisfied.

Following the running example of Figure 3.3, we assume that attributes T and R of *Sensor* form a foreign key for *Sensor2*. Moreover, assume that for each room and timestamp, the temperature readings of the 3 different sensors in that room depend conditionally on the (single) humidity reading for that timestamp (i.e., $H \rightarrow Tp$), and that this correlation pattern is quantified by the same CPT across all such groups of RVs. That can be expressed by defining 2 stripes, $S_{Tp} =< Sensor.Tp, \{\pi_{R,T,Sid} Sensor\} >$ for temperature and $S_H =< Sensor2.H, \{\pi_{R,T} Sensor2\} >$ for humidity, and a first order factor $\phi(S_{Tp}|S_H)$ which is made explicit through the one-to-many mapping:

$$\begin{aligned}(\forall \varepsilon_{Tp} \in S_{Tp}.\mathcal{E}, \forall \varepsilon_H \in S_H.\mathcal{E}) Pa(r.\varepsilon_{Tp}) = r.\varepsilon_H, \text{ s.t.} \\ \varepsilon_{Tp}.T = \varepsilon_H.T \wedge \varepsilon_{Tp}.R = \varepsilon_H.R\end{aligned} \tag{3.2}$$

The mapping $f[S_{Tp}, S_H]$ is graphically depicted in Figure 3.3(c). In this example we can see the application of Definition 3.3.3 in its full generality; each entity of the parent stripe S_H is associated with its 3 corresponding entities from the child stripe S_{Tp}, which can be uniquely defined because of the key-foreign key relationships that characterize the two entity sets.

We are now ready to formally define a first-order factor.

Definition 3.3.4 *[First-order Factor ϕ ($\{S_i\}$,$\{f_i\}$,CPT)] A first-order factor ϕ represents the conditional dependency of a child stripe S_c on a set of parent stripes $Pa(S_c) = \{S_{p_1}, \ldots, S_{p_n}\}$, ϕ $(S_c|\pi(S_c))$. It consists of:*

- *A child stripe S_c and a (possibly empty) ordered list of parent stripes $\{S_{p_1}, \ldots, S_{p_n}\}$.*

- *A (possibly empty) ordered list of stripe mappings $\{f_i[S_c, S_{p_i}]\}$, $(i = 1, \ldots, n)$, each one associating the entities of the child stripe with those of the corresponding parent stripe.*

- *A conditional probability table (CPT), quantifying the common local model, that holds for all the* RV *associations that are defined by the set of mappings $\{f_i[S_c, S_{p_i}]\}$, among the entities of the stripes involved, $P(S_c|\pi(S_c))$.*

First-order Bayesian Network: A BAYESSTORE FOBN model $\mathcal{M}_{FOBN}$ can now be defined as a set of first-order factors $\Phi = \{\phi_1, ..., \phi_n\}$, where each ϕ_i defines a local CPT model for the corresponding child stripe. Note that our definitions of stripes and stripe mappings are sufficient to guarantee that each individual FO factor can be grounded to a valid collection of local CPTs over propositional RVs (entities). In the presence of multiple FO factors (and possible connections across factors), ensuring that the global $\mathcal{M}_{FOBN}$ model can be grounded to a valid probabilistic distribution over entities is a little trickier. More specifically, note that even distinct stripes in factors can overlap (see Example 3 below), and additional structural constraints must be imposed on the model to guarantee grounding to an *acyclic* BN model.

An $\mathcal{M}_{FOBN}$ manages to abstract the conditional independence relationships represented among the RVs of a Bayesian Network, at the level of *populations* of RVs (stripes) that share the same local probabilistic model. By explicitly maintaining the association of each stripe with the RVs of the entities that comprise it, as well as the parent-child relationship between every pair of connected stripes (through the corresponding mapping function), a First-Order Bayesian Network $\mathcal{M}_{FOBN}$ can be invariably translated to a propositional Bayesian Network (BN), among the RVs that constitute the $\mathcal{M}_{FOBN}$, which become the nodes of the BN. Then, for every RV of the child stripe S_c of every first-order factor in $\mathcal{M}_{FOBN}$, an directed edge is added to the ground BN, which connects it with its corresponding parent RVs, as specified by the set of mapping functions $f_i()$'s, associated with that first order factor.

Thus, the joint probability distribution of the RVs in BN, can be calculated from the $\mathcal{M}_{FOBN}$ as:

$$P(X_1, ..., X_{|\mathcal{R}|}) = \prod_{i=1}^{N} \prod_{j=1}^{|\phi_i.S_c.\mathcal{E}|} CPT(s_j|Pa(s_j)), \tag{3.3}$$

where N is the number of first-order factors, $\mathcal{R}$ the set of RVs among all the incomplete relations of the probabilistic database $\mathcal{DB}^p$, $|\phi_i.S_c.\mathcal{E}|$ is the number of entities in the i^{th}'s factor child stripe $\phi_i.S_c$, s_j is the RV corresponding to the j^{th} entity of $\phi_i.S_c$, and $\pi(s_j)$ the set of s_j's parent RVs, associated with s_j according to the set of mappings $\{\phi_i.f(S_c, S_p)\}$, $S_p \in \pi(S_c)$.

Of course, the grounding of an $\mathcal{M}_{FOBN}$ in its propositional BN is neither desired, nor required in many cases. Besides its concise representation, an $\mathcal{M}_{FOBN}$ has the potential of exponentially accelerating inference as well. Poole [58] has demonstrated that common inference algorithms for

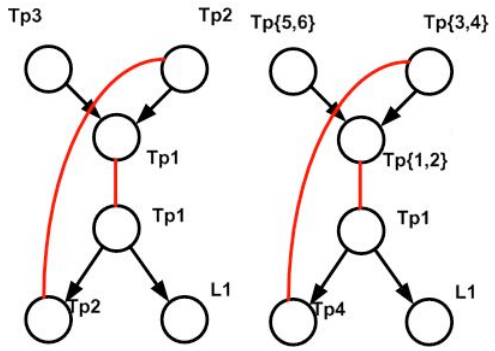


Figure 3.4. An invalid (on the left) and a valid (on the right) dependency graph of two First-Order Bayesian Networks

BNs can be adapted to work in a first-order setting. Thus, unless an efficient implementation of a first-order inference is not provided, the grounding process described above is not necessary.

One thing to note about $\mathcal{M}_{FOBN}$s is that, although they can be represented in a graphical structure as their propositional relatives (e.g. BNs, Markov Networks, etc.), each first order factor forms a disconnected component of that graph. This is in contrast to the equivalent propositional BN, where RVs are *explicitly* connected with all their parents through edges, forming large connected components. In an $\mathcal{M}_{FOBN}$ though, an entity may appear in the entity sets of multiple child and/or parent stripes, in different first-order factors, as each such factor captures different "correlation templates" (or first-order correlations as we call them) in which that entity participates. Thus, stripes in a $\mathcal{M}_{FOBN}$ can be *implicitly* correlated, because their entity sets may overlap, although in the graphical representation appear disconnected (see Figure 3.5). This by no means suggests a weakness of our formalism, but rather a unique property in the graphical modeling literature, that comes from the flexibility which the first-order specification of correlations has to offer!

Overlapping entity sets though can potentially cause the resulting $\mathcal{M}_{FOBN}$ to become invalid. In a propositional BN, the constraint that enforces the model's structural consistency with respect to the probabilistic distribution it represents is that the directed graph is acyclic. In an $\mathcal{M}_{FOBN}$, overlaps in the stripes' entity sets of the first-order factors may cause cyclic dependencies, which although are implicit since the subgraph of each first-order factor is disconnected from the rest, as we previously noted, these cycles become explicit as soon as the model is grounded to a propositional BN.

Since grounding is an expensive operation (proportional to the total number of RVs encoded by an $\mathcal{M}_{FOBN}$) the consistency of the latter can be verified on the first-order model's structure as well. To accomplish that, a *dependency graph* of the $\mathcal{M}_{FOBN}$ is constructed, which includes as nodes all the stripes (children and parents alike) from all the $\mathcal{M}_{FOBN}$'s first order factors, and two types of edges: directed, black edges, going from parent to child stripes, and *undirected*, red ones, between a parent and a child stripe of different factors, only if these stripes share entities between them.

If the dependency graph is acyclic, then the ground BN is guaranteed to be acyclic as well, and the $\mathcal{M}_{FOBN}$ represents a valid probability distribution. Basic condition for this quick test is that in each first-order factor, the entity sets of the child stripe with the ones of its parents *do not* overlap.

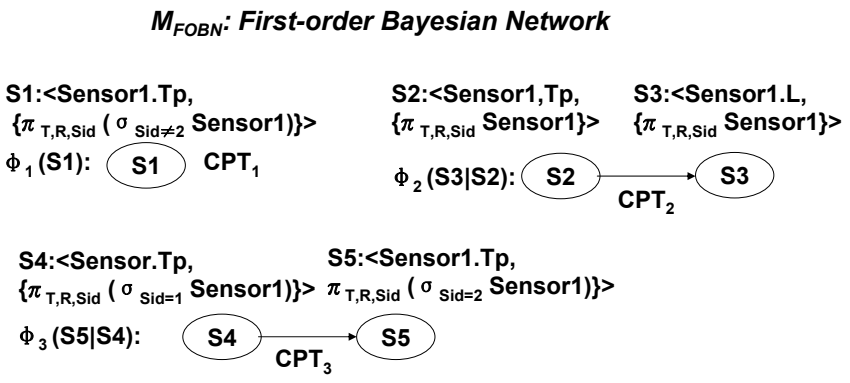


Figure 3.5. First-order Bayesian Network model over the incomplete relation *Sensor1* in Figure 3.3.

Nevertheless, a cycle does not necessarily mean that the model is inconsistent. Figure 3.4 depicts an inconsistent and a consistent dependency graph, both of which contain cycles. In the first one, by following the directed edges along the cycle, RV Tp_2 appears to be a parent of itself. In the second, where we implicitly assume that parent of Tp_1 is Tp_3, and Tp_2 is Tp_4, neither Tp_4 nor Tp_1, which appear in more than one nodes in the cycle, act as parents of themselves.

Following are three necessary conditions, which if they all hold, the $\mathcal{M}_{FOBN}$ in question is deemed *inconsistent*:

- The dependency graph contains at least one cycle, formed by alternating directed and undirected edges.
- If we identify the end points of the cycle as S_{start} (the beginning stripe) and S_{end} (the ending stripe), these stripes are by construction connected with an undirected edge, and thus contain some entities in common. Let this overlapping set of entities be S_O.
- By tracing back the lineage (parent-child mappings) of each entity in S_O, starting from S_{end}, and following the directed edges in reverse direction up to S_{start}, the path of entities should result in a cycle.

Example 3 *Figure 3.5 shows an example $\mathcal{M}_{FOBN}$ for our sensornet probabilistic database. There are three first-order factors: $\{\phi_1, \phi_2, \phi_3\}$. For ϕ_1, the child stripe $\phi_1.S_1$ is defined over the attribute Tp^p, and with an entity set of all the entities with $Sid \neq 2$. For ϕ_2, both the child and parent stripes are defined over the maximal entity set of the* Sensor1 *relation (i.e. for* ***all*** *tuples in* Sensor1*). Finally, for ϕ_3, the child stripe is defined over the attribute Tp^p of all the entities with $Sid = 2$, and the parent stripe is defined over Tp^p of all the entities with $Sid = 1$. In all cases, "1-1"*

strID	pos	token	Label
1	0	2181	
1	1	Shattuck	
1	2	North	
1	3	Berkeley	
1	4	CA	
1	5	USA	

Figure 3.6. An instance of the TOKENTBL table.

mapping functions are assumed, which along with the corresponding CPTs, are omitted from the figure in the interest of space.

Learning the shared correlation structure and the parameters of an FO probabilistic model from data is known to be a challenging task [64]. In a workshop paper [65], we discussed the complications of the learning process, and how it can be facilitated over a hierarchical version of FOBNs.

3.3.3 Summary

$\mathcal{M}_{FOBN}$ is a novel, declarative FO extension of BN models that is able to capture complex possible-worlds distributions of large PDB instances in a compact and scalable manner. Based on the solid probabilistic foundation of BNs, BAYESSTORE can model the complex correlation patterns present in real-world uncertain applications; in addition, through our novel FO extensions, such correlations can be declaratively expressed (and learned) at the appropriate level of granularity, with random variables (RVs) specified in a very flexible and schema-independent manner. (This is in contrast to PRMs [56, 59], where RVs correspond only to schema-level attributes; PRMs are a simple special case of the BAYESSTORE FO statistical model.)

3.4 BAYESSTORE Instance for Information Extraction

Although IE has been cited as a key driving application for probabilistic database research, there has been little work on effectively supporting the probabilistic IE process, storing the probabilistic IE outputs, and executing queries over them in a PDB. Gupta and Sarawagi's work [28] presented a mapping from probabilistic IE outputs to a PDB, but it showed that with a probabilistic database, which requires independent assumptions or supports only limited probabilistic correlations, the probability distribution over the possible extractions from an IE model can only be stored approximately.

In this section, we show how to bridge the gap between CRF-based IE and probabilistic databases, preserving the fidelity of the CRF distribution by storing both the text and the CRF model in BAYESSTORE. We describe an instance of the BAYESSTORE data model that can support

rich probabilistic IE models, such as conditional random fields (CRF) by (1) storing text-strings in an incomplete relation R as an inverted file with a probabilistic label^p attribute; and (2) representing the CRF model that encodes the probability distribution over all possible extractions. The BAYESSTORE representation of the CRF model uses the terminology from the BAYESSTORE representation of the FOBN model described in the previous section.

3.4.1 Token Table: The Incomplete Relation

The token table TOKENTBL is an incomplete relation R, which stores text-strings as relations in a database, in a manner akin to the inverted files commonly used in information retrieval. As shown in Fig. 3.6, each tuple in TOKENTBL records a unique occurrence of a token, which is identified by the text-string ID (strID) and the position (pos) the token is taken from. A TOKENTBL has the following schema:

TOKENTBL (strID, pos, token, label^p)

The TOKENTBL contains one probabilistic attribute – label^p, which can contain missing values, whose probability distribution can be computed from the CRF model. The deterministic attributes of the TOKENTBL are populated by parsing the input text-strings $\mathcal{D}$, with label values marked as missing by default.

3.4.2 Conditional Random Fields: The Probability Distribution

Unlike the case of a general FOBN or Markov Network, a linear-chain CRF model is a very restrictive version of a Markov Network. The only correlation factor $f(y_t, x_t, y_{t-1})$ is between the current label y_t, current token x_t and the previous label in the sequence y_{t-1}. Thus, the structure of the $\mathcal{M}_{CRF}$ model can be represented, using the terminologies in Section 3.3.2, by one first-order factor consists of three stripes $\{y_t, y_{t-1}, x_t\}$: $\Phi(\{y_t, y_{t-1}, x_t\}, f[y_t, y_{t-1}, x_t], \phi[y_t, y_{t-1}, x_t])$. Two out of the three stripes represent all the label attribute values, and the third stripe represents all the token attribute values in the TOKENTBL:

$$\begin{aligned} y_t &= <\text{TOKENTBL.Label}, \{\pi_{strID,pos}(\text{TOKENTBL})\} >, \\ y_{t-1} &= <\text{TOKENTBL.Label}, \{\pi_{strID,pos}(\text{TOKENTBL})\} >, \\ x_t &= <\text{TOKENTBL.Token}, \{\pi_{strID,pos}(\text{TOKENTBL})\} >, \end{aligned}$$

In directed graphical models such as FOBN, a mapping function $f_i()$ represent the mapping from the instances in a child stripe to the instances in its parent stripes. In undirected graphical models such as CRF, a mapping function represents the mapping between instances of stripes in a factor. The following function $f[y_t, y_{t-1}, x_t]$ represents the mapping between three stripes—the tokens x_t, the associating labels y_t, and the corresponding previous labels y_{t-1}.

$$f[y_t, y_{t-1}, x_t] = \{\sigma_{y_t.strID=y_{t-1}.strID=x_t.strID \wedge y_t.pos=y_{t-1}.pos+1=x_t.pos}(y_t, y_{t-1}, x_t)\},$$

Finally, the factor table $\phi[y_t, y_{t-1}, x_t]$ is the local probabilistic distribution between the label y_t and the previous label y_{t-1} given a token x_t.

3.5 Summary

In this chapter, we presented the data model of BAYESSTORE, which is comprised of a set of incomplete relations and a probability distribution over those relations encoded in a probabilistic graphical model. Such a data model manages to fully expose both the uncertainties in data and their probabilistic distribution as a unified and robust probabilistic representation that can be jointly manipulated. We showed instances of the BAYESSTORE data model for two different applications—sensor networks and information extraction over text—using two types of probabilistic graphical models. The FOBN model is general but complex, while the linear-chain CRF model is simple but limited. These two examples represent a wide spectrum of applications and probabilistic models we can support in BAYESSTORE.

Chapter 4

Probabilistic Relational Operators

Having presented the BAYESSTORE data model in Chapter 3, in this chapter, we describe algorithms for SPJ (i.e., selection, projection, join) operators over incomplete relations and their associated probability distribution. We use the FOBN model applied to sensor network data as the driving example. In the experimental section 4.5, we show effectiveness of various probabilistic selection algorithms in filtering out zero-probability tuples to reduce the input to the following expensive inference operators. We also conduct one preliminary experiment to show the best-case inference runtime improvement that can be achieved by exploiting first-order structures in FOBN model.

4.1 Overview

The standard algorithms for relational SPJ operators can only operate over deterministic relations. For probabilistic data analysis, we need to support probabilistic SPJ operators over a $\mathcal{DB}^p$ such as BAYESSTORE. A naive query processing approach is to (1) perform traditional relational operations on the set Ω of all possible worlds; (2) compute a new set of possible worlds Ω'; and (3) correctly map probabilities from F to F'. However, this computation is clearly intractable given the exponential number of possible worlds. Thus, the core challenge is to generate the new state of $\mathcal{DB}^p$ given a query without having to enumerate and process over all possible worlds.

We present new SPJ algorithms that seamlessly integrate the state-of-the-art SML techniques with relational query processing to support probabilistic SPJ queries and probabilistic model manipulations inside BAYESSTORE. The probabilistic SPJ algorithms in BAYESSTORE operate on both the data in the incomplete relations and the probability distribution in the PGMs. Operations on the PGM model are performed first in order to ensure that the resulting probability distribution F', encoded by the modified PGM, is compatible with the new state of $\mathcal{DB}^p$. Operations on the incomplete relations are performed second to filter out non-existent tuples (i.e., with zero existence probability) by exploiting the structure of the PGM. The second data filtering step is performed without impairing the soundless or accuracy of the result.

The following sections describe the algorithms for probabilistic selection, projection, and join over both the incomplete relations (i.e., sensor tables) and the probabilistic distribution encoded in a first-order Bayesian network.

4.2 Selection

We focus our discussion on the use of a selection predicate ϱ, that is a conjunction of atomic predicates ($\varrho = \wedge\varrho_i(i = 1, ..n)$), whose operators involve arithmetic comparisons $\Theta \in \{<, >, \leqslant, \geqslant, =\}$ between a probabilistic or deterministic attribute and a constant ($\varrho_i = A_i \Theta ct_i$). BAYESSTORE also supports other forms of atomic predicates, such as $\varrho_i = A_i \Theta B_i$ presented in Section 4.4, where probabilistic joins are discussed.

The techniques for performing selection *over the model* (as we call the $\mathcal{M}_{FOBN}$ modification process), which are discussed in Section 4.2.1, have proven to be quite fundamental for all the relational operations described in this chapter. On the other hand, selection over the tuples of R is not a trivial process either, as the filtering is no longer deterministic for probabilistic attributes. Section 4.2.2 introduces a basic algorithm for selecting incomplete data tuples, as well as two optimizations, which attempt to reduce the size of the resulting incomplete relation R', without affecting the validity of the selection output.

4.2.1 Selection over Model $\mathcal{M}_{FOBN}$

In a deterministic DB, selection over a relation R generates a new relation that contains only the tuples that satisfy the selection predicate. Should R be an incomplete relation (e.g., the *Sensor1* relation of Figure 3.3, in which some temperature and light readings are missing), a selection with predicate ϱ : (*Sensor1*.L^p = `Brt`), apart from removing the tuples that have a light value different than `Brt`, affects the distribution of possible worlds F as well, since worlds where $L =$ `Drk` cannot be generated by ϱ.

In statistical model terms, this probabilistic selection operation resembles that of computing the conditional distribution $\mathsf{Pr}[X|Y_1 = y_1, \ldots, Y_n = y_n]$ of a set of RVs X, given that RVs $\{Y_i\}_1^n \in X$, have specific values $\{y_1, \ldots, y_n\}$. Thus, it seems tempting to calculate the new possible-worlds distribution F' over the set of example RVs $X_{Sensor} = \{Tp_1, \ldots, Tp_n, L_1, \ldots, L_n\}$, by conditioning on all the RVs that correspond to *Sensor1*.L = `Brt`. Unfortunately, this operation does not result in the correct possible worlds distribution: In a nutshell, the problem here is that conditioning (and other standard model manipulations, such as marginalization) cannot by themselves express possible worlds with different numbers of tuples — even under the conditioned/marginalized model, the number of tuples in each possible completion of an incomplete relation R is going to be the same (i.e., $|R|$). In contrast, applying the selection operation over the possible worlds *can obviously result in worlds with different numbers of tuples*, by filtering out tuples that do not satisfy the selection predicate; essentially, probabilistic selection introduces *tuple-existence uncertainty* (i.e., the *existence* of a tuple in a possible world becomes *uncertain*).

Continuing with our example, a given tuple $t \in$ *Sensor1* appears only in the possible worlds where the attribute $L =$ `Brt`, and in all the rest it is filtered out completely. On the other hand,

by simply calculating the conditional $\mathsf{Pr}[X_{Sensor}|L_1 = \texttt{Brt}, \ldots, L_n = \texttt{Brt}]$ directly over the model, the possible worlds corresponding to the resulting joint pdf all have *the same* number of tuples (i.e., the number of tuples in *Sensor1* with $L \neq \texttt{Drk}$) all having $L = \texttt{Brt}$. (It should also be intuitively clear that this conjunctive conditioning does not have the right semantics for our selection operation.) The fact that the *cardinality* of tuples between possible worlds can vary cannot be directly expressed by standard conditioning/marginalization operations on the model.

We capture tuple existence uncertainty by introducing the probabilistic attribute $Exist^p$ in the schema of the incomplete relation being selected, if it is not contained already. Depending on whether the selection predicate ϱ involves a deterministic A^d, or a probabilistic attribute A^p, the corresponding selection over a model $\sigma(\mathcal{M}_{FOBN})$ behaves differently.

The atomic predicate $\varrho : (A^d \Theta ct)$, restricts the entity set to which the predicate ϱ applies. For example, $\varrho : (Sid = 3 \wedge Tp = \texttt{Hot})$ restricts the entity set of predicate ϱ to only include the entities with Sid equals 3. Entities with $Sid \neq 3$ cannot exist in the resulting relation (i.e., their $Exist^p$=0).

Existence uncertainty is introduced in the model when ϱ involves a probabilistic attribute $\mathcal{A}^p$. For example, if a predicate $Tp = \texttt{Hot}$ is applied to a tuple t with a missing Tp value, the presence of t in the output becomes probabilistically dependent on $t.Tp$'s value. Intuitively, this operation corresponds to changing the local model of the factor that corresponds to R's $Exist^p$ attribute, to be dependent on that probabilistic attribute (for our example, Tp).

Figure 4.1 shows the `select-model` algorithm, based on a predicate ϱ of the form $A^d \Theta_1 ct_1 \wedge A^p \Theta_2 ct_2$. For example, $\varrho = (Sid = 3 \wedge Tp = \texttt{Hot})$. The algorithm can be easily extended to process a conjunctive predicate ϱ with any number of atomic predicates. (Join predicates (e.g., $A \Theta B$) are discussed in Section 4.4.)

Lines 1-2 define the entity set on which the selection predicate applies: $\varrho.\mathcal{E}$. The presence of the deterministic attribute inside ϱ ($Sid = 3$) essentially restricts its entity set $\varrho.\mathcal{E}$ (e.g., to all tuples with $Sid = 3$); otherwise, the predicate applies to any entity within R's entity set.

If $\varrho.\mathcal{E}$ is non-empty, a new first-order factor ϕ is added to the model $\mathcal{M}_{FOBN}$ (lines 5-11). ϕ represents the local model of an $Exist^p$ attribute stripe over the entity set $\varrho.\mathcal{E}$: $S_c < Exist^p, \varrho.\mathcal{E} >$. $A^p \Theta_2 ct_2$ determines the parent stripe, $\phi.S_p$, of S_c. The latter is defined over the probabilistic attribute A^p in ϱ (e.g., a stripe on attribute Tp^p becomes a parent of a stripe on the $Exist^p$ attribute, with the same entity set, which in this example contains all the tuples with $Sid = 3$). A one-to-one mapping is established between $\phi.S_c$ and its parent stripe $\phi.S_p$, in line 8 (e.g., the value of the $Exist^p$ attribute in entity ε maps to the value of Tp^p in the same ε). Finally, the CPT table of ϕ is set to represent the deterministic conditional distribution: $Exist^p = 1$ with probability 1, iff $A^p \Theta_2 ct_2 = true$.

Lastly, lines 12-16 deal with the residual entity set (e.g., the entities with $Sid \neq 3$), which contains all the entities of R that are not included in $\varrho.\mathcal{E}$: ($R.\mathcal{E}$- $\varrho.\mathcal{E}$). A second first-order factor ϕ is defined over the $Exist^p$ attribute of the residual entity set, which specifies that none of these entities exists.

In the interest of space, the pseudocode of Figure 4.1 omits a preprocessing step that examines if there exist first-order factors in $\mathcal{M}_{FOBN}$ that are defined over attribute $Exist^p$. Such a case

SELECT-MODEL $(R, \mathcal{M}_{FOBN}, \varrho = A^d\Theta_1 ct_1 \wedge A^p\Theta_2 ct_2)$

1 $\mathcal{M}_{FOBN}$' $\leftarrow \mathcal{M}_{FOBN}$

2 $\varrho.\mathcal{E} \leftarrow < A^d\Theta_1 ct_1 >$

3 *// Modification of pre-existing factors* $\phi \in \mathcal{M}_{FOBN}$ *if* $\phi.S_c$

4 *// over Exist^p – Step omitted due to space constraints.*

5 **if** $\varrho.\mathcal{E} \neq \emptyset$ **then**

6 $\quad \phi.S_c \leftarrow < R.Exist^p, \varrho.\mathcal{E} >$

7 $\quad \phi.S_p \leftarrow < R.A^p, \varrho.\mathcal{E} >$

8 $\quad \phi.f \leftarrow$ *a one-to-one mapping between* S_p *and* S_c

9 $\quad \phi.CPT \leftarrow \{Exist^p = 1$ *iff* $\varrho.A^p\Theta_2 ct_2 ==$ *true*$\}$

10 $\quad \mathcal{M}_{FOBN}$' $\leftarrow \mathcal{M}_{FOBN}$' $\cup \phi$

11 **endif**

12 **if** $\varrho.\mathcal{E} \neq R.\mathcal{E}$ **then**

13 $\quad \phi.S_c \leftarrow < R.Exist^p, (R.\mathcal{E} - \varrho.\mathcal{E}) >$

14 $\quad \phi.CPT \leftarrow \{Exist^p = 0\}$

15 $\quad \mathcal{M}_{FOBN}$' $\leftarrow \mathcal{M}_{FOBN}$' $\cup \phi$

16 **endif**

17 **return** $\mathcal{M}_{FOBN}$'

Figure 4.1. Algorithm for selection over model with predicate $\varrho = (A^d\Theta_1 ct_1 \wedge A^p\Theta_2 ct_2)$

might arise if R is a result of a previous selection. For such an existence factor ϕ, the entity sets of its stripes are restricted to contain the entities that lie in the intersection of $\phi.S_c.\mathcal{E}$ and $\varrho.\mathcal{E}$. The correctness of the final possible-worlds distribution, follows trivially from the existence RVs added to the FOBN model.

Example 4 *As an example, suppose the selection predicate of* $\sigma_{Sid=3 \wedge Tp^p=\texttt{Hot}}(Sensor)$ *is applied on the model* $\mathcal{M}_{FOBN}$ *in Figure 3.5. The selection operation first extends the schema of* Sensor1 *with a probabilistic attribute* $Exist^p$*, which represents the existence uncertainty of every tuple in* Sensor1*. It then modifies the model by adding two first-order factors over two different stripes of the* $Exist^p$ *attribute. The first specifies that for all entities with* $Sid = 3$*,* $Exist^p$*'s value is dependent on* Tp^p*'s value: if* $Tp^p = \texttt{Hot}$ *then* $Exist^p = 1$*, and 0 otherwise. The second specifies that for all entities with* $Sid \neq 3$*,* $Exist^p = 0$*.*

4.2.2 Selection over an Incomplete Relation $\sigma(R)$

In a traditional deterministic relation, a selection with predicate ϱ filters out exactly those tuples that do not satisfy the predicate. For an incomplete relation R, such a naive filtering operation would produce incorrect results. The first problem lies in whether to select the tuples with missing values in the predicate attribute: if the missing value has zero probability to satisfy the predicate, then the tuple should be filtered. A conservative approach would select all tuples with missing values. The second problem is that there may be tuples in R that do not satisfy the predicate, but are still relevant, due to their correlation with another tuple that has non-zero probability of satisfying the predicate. The absence of these *evidence* tuples will lead to incorrect probabilistic inference computations later on.

As an example, consider two sensors, one placed on the exterior of a building, the other placed in a heating duct. The sensors are anti-correlated: when the exterior sensor is cold, the duct sensor is more likely to be hot. A query that asks for the temperatures of all sensors detecting heat, must retain the $Cold$ temperature reported by the exterior sensor, if the duct's sensor temperature reading in a given tuple is missing.

Thus, selection over an incomplete relation $\sigma(R)$ has to be *sufficient*, in retaining evidence tuples in R, which are relevant to any tuple that may satisfy the predicate with non-zero probability. On the other hand, selection should also be *minimal*, in deleting any tuple that neither satisfies the predicate nor is correlated with other "candidate" tuples. The "minimum" incomplete relation can be achieved by computing the marginal probability distribution over all the $Exist^p$ attribute values, and filtering all the tuples that have 0 probability of having $Exist^p = 1$. The complexity of this inference computation is known to be NP-hard, in general. We proceed to describe a base algorithm for $\sigma(R)$, and two optimizations that reduce the former's complexity, while being as economical in retaining evidence tuples as possible.

Base Algorithm. Figure 4.2 displays the basic version of the data selection algorithm. According to the two objectives stated earlier, it selects tuples which either satisfy the predicate or contain missing values (tuple set T), while retaining all the tuples that are correlated with at least one tuple from T. The latter are determined by computing a transitive closure operation over T, using a *tuple correlation graph* – an undirected graph in which nodes correspond to R's tuples, and undirected edges represent correlations between pairs of tuples ("vertical" correlations).

In particular, the data selection algorithm initially uses the input predicate ϱ to select the tuples in the incomplete relation R, which either contain evidence satisfying the predicate, or their probabilistic attribute value in the predicate is missing (line 1).

In lines 2-20, the transitive closure of the tuple set E_1 is computed, using semi-naive evaluation, over the tuple correlation graph. At the end of each iteration, a new set of tuples E_{i+1} is generated, containing the parents and children of each tuple $t \in E_i$, as defined in the correlation graph. At each iteration, the newly chased tuples are added to E_{corr}, the set of all correlated tuples with t.

Lines 7-16, represent the chasing of t's correlated tuples. All first-order factors ϕ in $\mathcal{M}_{FOBN}$ are traversed: if one of the *parent* stripes' entity sets $\phi.S_{p_j}.\mathcal{E}$ contains an entity that maps to t (or more formally, if the projection of t on $\phi.S_{p_j}.\mathcal{E}$'s schema belongs in $\phi.S_{p_j}.\mathcal{E}$), then the entity(ies) in the child stripe $\phi.S_c$ to which it maps, are computed through the reverse mapping $f_j^{-1}(t)$ and added to E_{corr}; if t can be associated with an entity of $\phi.S_c.\mathcal{E}$, then all its corresponding entities from each of $\phi.S_c$'s parents are chased as well.

Example 5 *Figure 4.3 depicts the model $\mathcal{M}_{FOBN}$ of* Sensor1*, and the latter's tuple correlation graph. Tuples $\{t_1, t_2\}, \{t_4, t_5\}$ indicate correlated pairs. As a first step, the algorithm selects tuples from* Sensor1 *which satisfy the predicate or contain missing values. Consequently, it computes the incomplete relation* Sensor1*', calculating the transitive closure of the tuples generated from the first step, given the tuple correlation graph. Note that in the latter step, t_2 is added, because even though it does not satisfy the predicate, it has a vertical correlation with t_1.*

Evidence-Based Early-Stopping Optimization. The transitive closure computation that the base algorithm utilized to collect all vertically correlated tuples, despite its correctness, might conservatively select more tuples than necessary. However, two tuples t_1 and t_2 may be *conditionally independent*, even if there exists a vertical correlation between them. Consequently, we can accelerate the computation, by stopping early when encountering conditionally independent tuples, and thus reduce the number of resulting tuples.

Identifying conditionally independent tuples is done by means of the "Bayes Ball" algorithm [16]. Bayes Ball is a standard Graphical Model algorithm that takes as input a Bayesian Network and evidence values for some of its random variables, and determines pairs of nodes in the network that are independent, given the evidence. The details of the Bayes Ball algorithm can be found in [16]. Figure 4.4 modifies our `SelPlain` algorithm by replacing the transitive closure

SELECT-DATA-BASE $(R, \mathcal{M}_{FOBN}, \varrho)$

1 $S \leftarrow$ SELECT(R, ϱ); $R' \leftarrow \emptyset$; $E_1 \leftarrow S$; $i \leftarrow 1$;
2 **while** $E_i \neq \emptyset$ **do**
3 $E_{i+1} \leftarrow \emptyset$;
4 **for** *each* $t \in E_i$ **do**
5 $R' \leftarrow R' \cup t$; $E_{corr} \leftarrow \emptyset$;
6 *// Locate correlated tuples to t by traversing* $\mathcal{M}_{FOBN}$
7 **for** *each* $\phi \in \mathcal{M}_{FOBN}$ **do**
8 *// If t is associated with an entity in one of* ϕ*'s*
9 *// parent stripes...*
10 **if** $\exists j (j \in \{1, \ldots, n\}) : \ t \in \phi.S_{p_j}.\mathcal{E}$ **then**
11 $E_{corr} \leftarrow E_{corr} \cup f_j^{-1}(t)$;
12 *// or child stripe...*
13 **else if** $t \in \phi.S_c.\mathcal{E}$ **then**
14 **for** *each* $\phi.f_j$ **do**
15 $E_{corr} \leftarrow E_{corr} \cup f_j(t)$;
16 **endfor endif endfor**
17 $E_{i+1} \leftarrow E_{i+1} \cup E_{corr}$;
18 **endfor**
19 $i \leftarrow i + 1$;
20 **endwhile**
21 **return** R'

Figure 4.2. Base Algorithm for select-over-data

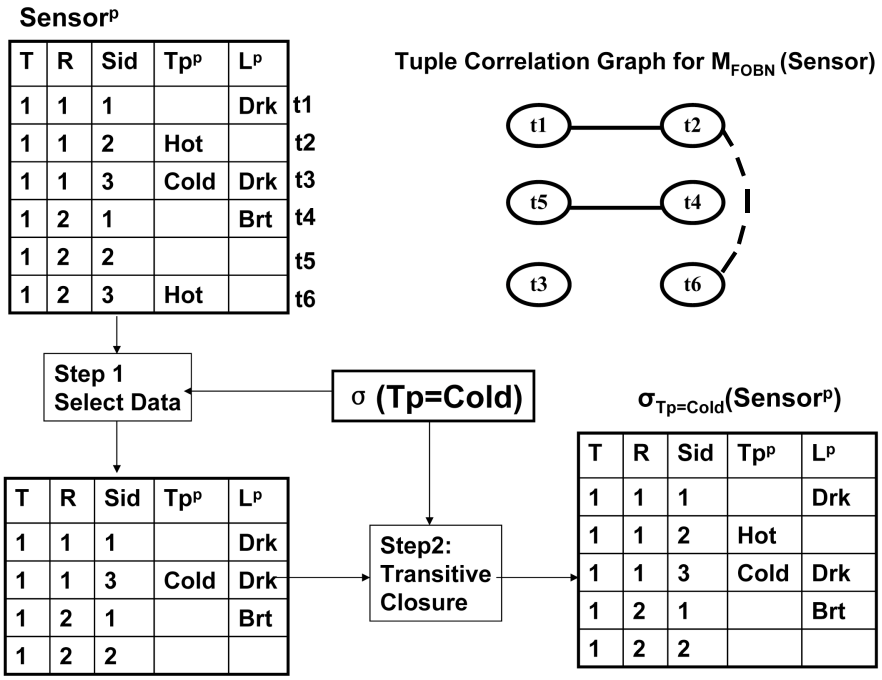


Figure 4.3. An example illustrating the Base Select-over-data Algorithm for $\sigma_{Tp='Cold'}(Sensor)$.

operation with the Bayes Ball algorithm to avoid retaining tuples that, given the evidence, are independent of any tuples that may be in the output.

Example 6 *Continuing our example from Figure 4.3, let us suppose that another vertical correlation exists between tuples t_2 and t_6 ($t_2 \rightarrow t_6$), which is reflected with a dotted line in the tuple correlation graph. Although t_6 is correlated with t_2, the Bayes Ball algorithm indicates that it is conditionally independent with t_1, and therefore, the transitive closure stops at t_2; t_6 doesn't need to be chased.*

Factor-Based Data Filtering Optimization. Recall that the base algorithm conservatively maintained all the tuples that contained a missing value for the attribute mentioned in ϱ, assuming that the probability for such a tuple to have a value that satisfies ϱ is non-zero. Nevertheless, we could have used the $\mathcal{M}_{FOBN}$ model, conditioning on the evidence present in R, to verify if this holds for every such tuple. In particular, we have not yet utilized the probability distribution encoded in $\mathcal{M}_{FOBN}$, to actually verify if the probability of satisfying the predicate is non-zero, conditioning on the evidence present in the incomplete relation.

As an example, let us examine the following query over the *Sensor1* relation of Figure 4.3: $\sigma_{L=\texttt{Drk}(Sensor)}$. Suppose the local model of L, ϕ_2, has a conditional probability table ϕ_2.CPT, indicating that probability of $L = \texttt{Drk}$ is zero when $Tp = \texttt{Hot}$. Thus t_2, even with its L value missing, has zero probability satisfying the predicate.

EVIDENCESEL $(R, \mathcal{M}_{FOBN}, \varrho)$

1 $S \leftarrow$ SELECT(R, ϱ); $R' \leftarrow \emptyset$; $E_1 \leftarrow S$;
2 **while** $E_i \neq \emptyset$ **do**
3 $E_{i+1} \leftarrow \emptyset$
4 **for** *each* $t \in E_i$ **do**
5 $R' \leftarrow R' \cup t$;
6 *// Locate tuples correlated with t, by calling Bayes-Ball*
7 *// algorithm over model* $\mathcal{M}_{FOBN}$
8 $E_{corr} \leftarrow$ BAYES-BALL$(t, \mathcal{M}_{FOBN})$
9 $E_{i+1} \leftarrow E_{i+1} \cup E_{corr}$
10 **endfor**
11 $i \leftarrow i + 1$
12 **endwhile**
13 **return**R'

Figure 4.4. EvidenceSel algorithm for selection over data based on evidence

Since probabilistic inference is an expensive operation, as a relaxation, we avoid computing the conditional probability of the RV of *every* missing value, given all the available evidence in R and ϱ. Instead, we focus on tuples with RVs that are involved only in horizontal correlations. In particular, for each RV r corresponding to such tuples, we do not calculate the conditional over the whole model, $\mathsf{Pr}[r|X_{ev} \cup \varrho.A^p = ct]$, where X_{ev} represents the vector of evidence data in R, but rather we calculate $\mathsf{Pr}[r|\overrightarrow{Y}_{ev} \cup \varrho.A^p = ct]$, where $\overrightarrow{Y}_{ev}$ is the vector of RVs from the same tuple as r, instantiated with the evidence in the tuple. This probability can be obtained from the CPT of the horizontal factor ϕ in which r belongs (as part of $\phi.S_c$). Should $\mathsf{Pr}[r|\overrightarrow{Y}_{ev} \cup \varrho.A^p = ct] = 0$, the tuple can be safely removed from the result set, and not chased for other correlated tuples to it.

4.3 Projection

In this section, we concentrate on projection $\pi_{\Pi}(R)$ without duplicate elimination, where the projection list $\Pi = \{A_1, .., A_n\}$ contains the primary key. This assumption guarantees that there will be no duplicates generated from the project operation, because all the primary key attributes remain in the resulting relation.

The inherent difficulty with duplicate elimination lies in the fact that it is inherently not a first-order operation. To be able to account for duplicates, we would have to model the uncertain quantities of the probabilistic DB *at the tuple level*, which would require that we ground the first order model associated with the DB. Exploring more concise probabilistic representations to model duplicates is left as future work.

Traditional projection semantics over a deterministic relation, would result in retrieving each tuple from the relation, while keeping only a subset of its attributes. Such an operation over an incomplete relation though would generate incorrect results, because deleting the attributes that are not in the subset A may lead to loss of evidence, which is needed to generate the correct probabilistic distribution function F.

Not unlike selection, projection over an incomplete relation R retains some attributes A^* that are not in the projection list Π, if they are *correlated* with other attributes in Π, and deletes only the uncorrelated attributes $B = Sch(R) - \Pi - A^*$, where $Sch(R)$ is the schema (attribute list) of R. The project operation over the model $\mathcal{M}_{FOBN}$ deletes the first-order factors that involve attributes in the set B only.

The project algorithm is shown in Figure 4.5. In lines 1-5, the algorithm traverses all first-order factors ϕ in the model $\mathcal{M}_{FOBN}$. If ϕ involves two attributes A^p and B^p, where A^p belongs in the project list ($A^p \in \Pi$), and B^p does not ($B^p \notin \Pi$), then A^p and B^p are correlated by factor ϕ. Thus, on line 3, the algorithm retains B^p in project attribute set Π. In lines 6-9, the algorithm traverses all the first-order factors a second time, and processes the model $\mathcal{M}_{FOBN}$' to remove first-order factors that only involve attributes which are not in the newly computed set Π. In lines 11-13, the algorithm iterates through every tuple in the incomplete relation R, and performs a traditional projection of each tuple on the attributes that remained in Π.

Example 7 *Using the example in Figure 4.3, suppose we have a new probabilistic attribute Hu-*

PROJECT $(R, \mathcal{M}_{FOBN}, \Pi)$

1 $\mathcal{M}_{FOBN}$' $\leftarrow \mathcal{M}_{FOBN}$

2 **for** *each* $\phi \in \mathcal{M}_{FOBN}$ **do**

3 **if** $\exists i, j : \ \phi.S_i.A^p \in \Pi \ \&\& \ \phi.S_j.B^p \notin \Pi$ **then**

4 $\Pi \leftarrow \Pi \cup B^p$

5 **endif endfor**

6 **for** *each* $\phi \in \mathcal{M}_{FOBN}$ **do**

7 **if** $\forall i, \ \phi.S_i.A^p \notin \Pi$ **then**

8 $\mathcal{M}_{FOBN}$' $\leftarrow \mathcal{M}_{FOBN}$' $- \phi$;

9 **endif endfor**

10 $R' \leftarrow \emptyset$

11 **for** *each* $t \in R$ **do**

12 $R' \leftarrow R' \cup t(\Pi)$

13 **endfor**

14 **return** $R', \mathcal{M}_{FOBN}$'

Figure 4.5. Projection algorithm over Incomplete Relations

midity H, and a new first-order factor ϕ_4, which represents a prior probability table for all attribute values in H. Suppose we are to perform the projection $\pi_{\{T,R,Sid,L\}}$ on this modified incomplete relation Sensor1*. Since the first-order factor ϕ_2 encodes a correlation between attributes Tp and L, Tp is then included in the projection attribute set Π. Attribute H is not included in Π, because it is not correlated with any attribute in Π. Thus, the model of* Sensor1 *is modified by deleting ϕ_4, because it only involves attribute H. Finally, the new incomplete relation is generated by projecting every tuple of* Sensor1 *on the subset of attributes $\Pi = \{T, R, Sid, Tp, L\}$.*

4.4 Join

We now turn our attention to binary selection predicates. In particular, we will examine how the join operator ($\bowtie_\varrho$) between two incomplete relations R_1 and R_2, whose entities' correlation patterns are captured by the first order model $\mathcal{M}_{FOBN}$, determines the contents of the resulting incomplete relation $R = R_1 \bowtie_\varrho R_2$, as well as the necessary modifications to $\mathcal{M}_{FOBN}$. We will concentrate on join predicates of the form: $\varrho = (R_1.A^p \Theta R_2.B^p)$, where attributes A and B are probabilistic.

4.4.1 Join over Model $\mathcal{M}_{FOBN}$

The modifications to $\mathcal{M}_{FOBN}$ which are required to maintain the consistency of the probability distribution F with respect to R, are quite similar to those that the selection over the model algorithm performs. As in Section 4.2.1, we need to capture the *existence uncertainty* of every joinable pair of tuples $r_1 \in R_1$ and $r_2 \in R_2$, in the final result. The possible universe of these pairs corresponds to $R_1 \times R_2$. Nevertheless, these possibilities are restricted by the join predicate ϱ. Hence, the challenge, as in selection, lies in capturing the existence uncertainty of all *probable* tuple pair combinations in a *first order* fashion.

To accomplish this, we extend $\mathcal{M}_{FOBN}$ by adding a first-order factor over $R.Exist^p$, which depends on the probabilistic attributes A^p and B^p that participate in ϱ. The algorithm is described in Figure 4.6. Initially, as a preprocessing step (lines 4–8), we rename the attributes in the schema of R, $Sch(R)$, to be able to refer to each attribute of R by a *unique* name. This renaming step needs to be carried over the attributes of the stripes of each first-order factor in $\mathcal{M}_{FOBN}$ as well, so that the correspondence of $\mathcal{M}_{FOBN}$'s stripes with the probabilistic attributes of R is maintained.

Lines 10–17 define the new existence first-order factor ϕ. Essentially, the entity set of ϕ's child stripe, $\phi.S_c.\mathcal{E}$ corresponds to the cross product of the entities of the relations to be joined, $R_1(\overline{K}, \ldots, A^p)$ and $R_2(\overline{K}, \ldots, B^p)$, where $\overline{K}$ and $\overline{K}'$ are the sets of attributes comprising the primary keys of R_1 and R_2 respectively, as indicated by line 2. $\phi.S_c$'s parents are the stripes $\phi.S_{p_1}$ and $\phi.S_{p_2}$, representing the populations of RVs for the join attributes $R_1.A^p$ and $R_1.B^p$. Each entity ε_c from $\phi.S_c.\mathcal{E}$ is associated with the original "copies" of the two entities, ε_1 and ε_2, from which it was generated – one from each respective parent stripe. This many-to-one association

between an entity of one of the two initial relations, and its multiple copies in the entity set of $\phi.S_c$, is captured by the mapping functions $\phi.f_1$ and $\phi.f_2$ (lines 14–15). Finally, the CPT of this new factor represents the deterministic conditional distribution $Exist^p = 1$ with probability 1, iff $R_1.A^p \Theta R_2.B^p = true$.

As a special case, if R_1 or R_2 are a result of a previous selection, in which case $\mathcal{M}_{FOBN}$ contains existence factor(s) already, a similar process takes place, which extends the entity sets of the child and parent stripes involved, as well as the condition defining the factors' CPTs, to the resulting cross-product domain of the join. The only difference there is the formulation of the local model of the existence random variables $Exist^p$ in the resulting join model. The algorithm generates the local model of $Exist^p$ of the join result R as a conjunction ($\bigwedge$) of the condition in the local model of ***R1***.$Exist^p$, the condition in the local model of ***R2***.$Exist^p$, and the join condition.

4.4.2 Joining Incomplete Relations $\sigma(R)$

As in the case of selection, the presence of missing values for the probabilistic attributes of an incomplete relation, causes complications when the latter is to participate in a join. In particular, when the attributes participating in a join predicate are probabilistic ($R_1.A^p \Theta R_2.B^p$), each tuple of R_1 (equivalently for R_2) that has a missing value for A^p, may satisfy ϱ and thus participate in the join result, depending on the possible values it can take, according to the $\mathcal{M}_{FOBN}$ and the evidence present in R_1 and R_2. Following the same intuition as in Section 4.2.2, all such tuples from both relations should be considered "joinable" *conservatively*, as the join operation per se becomes uncertain.

In addition, we might also need to include in the result the join of two tuples $r_1 \in R_1$ and $r_2 \in R_2$ that do not satisfy the join predicate, for the same reason that this problem arises in selection: this resulting tuple might be correlated with another tuple from $R = R_1 \bowtie_\varrho R_2$ that satisfies ϱ, and hence its absence from R can erroneously interfere with probabilistic inference operators. Therefore, the set of these seemingly disqualified join tuples needs to be transitively calculated from the tuples of R' that satisfy ϱ, using the probabilistic information provided in $\mathcal{M}_{FOBN}$.

In essence, the above can be phrased as a direct extension of the selection over a single incomplete relation, studied in section 4.2.2. Taking advantage of the techniques developed there, as well as common Relational Algebra equivalences, we first compute the cross product of the relations to be joined, on top of which we apply our probabilistic selection operator, using the same predicate as in ϱ: $R_1 \bowtie_\varrho R_2 = \sigma_{\varrho=(R_1.A \Theta R_2.B)}(R_1 \times R_2)$.

Example 8 *Given the example in Figure 4.7, the query we want to evaluate over these two incomplete relations is:*

```
SELECT *  FROM Sensor1 R1, Sensor2 R2 WHERE R1.L = R2.L
```

The algorithm initially renames the attributes in the definition of each factor's stripe S_i, so that they coincide with the resulting relation's schema: $R(\underline{Sid1}, \underline{Sid2}, Tp^p, L1^p, L2^p, Exist^p)$. A de-

JOIN $(R_1(\overline{K}, \ldots, A^p), R_2(\overline{K}', \ldots, B^p), \mathcal{M}_{FOBN}, \varrho)$

1 $R \leftarrow R_1 \bowtie_{\varrho} R_2$;

2 $\varrho.\mathcal{E} \leftarrow \pi_{\overline{K} \cup \overline{K}'}(R_1 \times R_2)$

3 *// Assign unique names to the stripes of the factors in* $\mathcal{M}_{FOBN}$

4 **for** *each* $\phi \in \mathcal{M}_{FOBN}$ **do**

5 $\quad \phi.S_c \leftarrow$ RENAME$(\phi.S_c, Sch(R))$

6 $\quad$ **for** *each parent stripe* $\phi.S_{p_i},\ i = \{1, \ldots, n\}$ **do**

7 $\quad\quad \phi.S_{p_i} \leftarrow$ RENAME$(\phi.S_{p_i}, Sch(R))$

8 **endfor endfor**

9

10 *// Create an existence factor*

11 $\phi.S_c \leftarrow < R.Exist^p, \varrho.\mathcal{E} >$

12 $\phi.S_{p_1} \leftarrow < R_1.A^p, R_1.\mathcal{E} >$

13 $\phi.S_{p_2} \leftarrow < R_2.B^p, R_2.\mathcal{E} >$

14 $\phi.f_1 \leftarrow \forall \varepsilon_c \in \phi.S_c.\mathcal{E},\ \exists \varepsilon_1 \in \phi.S_{p_1}.\mathcal{E},\ \phi.f_1(\varepsilon_c) = \varepsilon_1 :\ \varepsilon_c.\overline{K} = \varepsilon_1.\overline{K}$

15 $\phi.f_2 \leftarrow \forall \varepsilon_c \in \phi.S_c.\mathcal{E},\ \exists \varepsilon_2 \in \phi.S_{p_2}.\mathcal{E},\ \phi.f_2(\varepsilon_c) = \varepsilon_2 :\ \varepsilon_c.\overline{K}' = \varepsilon_2.\overline{K}'$

16 $\phi.CPT \leftarrow \{Exist^p = 1$ *iff* $\varrho == true\}$

17 $\mathcal{M}'_{FOBN} \leftarrow \mathcal{M}_{FOBN} \cup \phi$

18 **return** $R, \mathcal{M}'_{FOBN}$

Figure 4.6. Join algorithm over Model $\mathcal{M}_{FOBN}$.

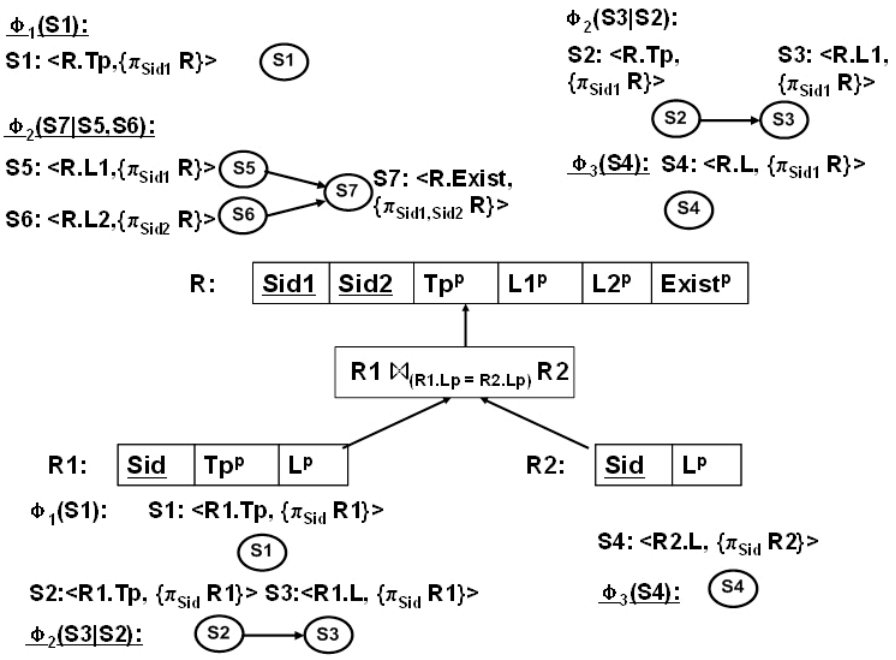


Figure 4.7. The join operation over the model of two incomplete relations, *Sensor*$_1$ and *Sensor*$_2$.

terministic first-order factor ϕ_4 *for* $R.Exist^p$ *is added to* $\mathcal{M}_{FOBN}$. *Its child stripe's entity set,* $S7.\mathcal{E}$ *is defined as:* $\{\pi_{Sid1,Sid2}R\}$, *while the parent stripes associated with it,* $S5$ *and* $S6$, *over the attributes* $R.L1$ *and* $R.L2$ *included in the join predicate, have entity sets:* $\{\pi_{Sid1}R\}$ *and* $\{\pi_{Sid2}R\}$ *respectively. Each entity in the parent stripe* $S5.\mathcal{E}$ *(likewise for* $S6$*) has a unique* $Sid1$ *value,* $s1$, *which is mapped with a set of children entities from* $S.\mathcal{E}$, *all having* $Sid1 = s1$ *and* $Sid2 \in$ $\mathsf{dom}(Sid2)$. *Finally, the CPT of* ϕ_4 *is defined as:*$(R.Exist^p = 1$, *iff* $R.L1 = R.L2)$.

4.5 Experimental Evaluation

We have implemented and evaluated various algorithms for probabilistic selection presented in Section 4.2. We have also implemented a very limited first-order inference optimization over FOBNs to show the best-case runtime improvement by exploiting first-order structures. The main findings of our experimental study can be summarized as follows:

- In our experiment setup, we generate selection predicates with different selectivity, data with different percentage of missing values, and model with different correlations. The experimental results consistently show that, compared to the strawman algorithm, the proposed

selection algorithm can drastically reduce the size of the output without affecting the accuracy of the result. We believe that this experiment result holds in general for other applications and models. We also show that this reduction in the input to the subsequent inference operator can decrease inference runtime by up to 5 times in our experimental setup.

- We conduct one preliminary experiment to show the best-case inference runtime improvement that can be achieved by exploiting first-order structures in FOBN model. We also acknowledge cases where the overhead can outweigh the improvement of first-order inference in runtime. An optimizer is needed to be able to choose between different inference algorithms and optimizations. In Chapter 7, we show our effort in that direction.

Our implementation is based on Postgres 8.2.4 (`www.postgresql.org`) and uses Murphy's BNT Toolbox (`www.cs.ubc.ca/~murphyk/Bayes/bnt.html`) for inference. In this section we describe the results of our experimental evaluation, which focused primarily on selection algorithms. All experiments were run on a 2.8Gz CPU, 1GB RAM system running the Fedora Linux Core 5.

4.5.1 Methodology

For our evaluation we used an extension of the example scenario of Section 3.3.1—a hotel with 100 rooms, each instrumented with 8 environmental sensors, measuring temperature and light over time. All the sensor readings are stored in an incomplete relation *Sensor*. The distribution F of this probabilistic database is captured by a FOBN model $\mathcal{M}_{FOBN}$. The $\mathcal{M}_{FOBN}$ contains 4 first-order factors: a prior for all temperature readings of the $1^{st}, 3^{rd}$ and 5^{th} – 8^{th} sensor for every room $\phi(Tp)$, one capturing the same correlation pattern between the light and temperature readings over every sensor in the hotel $\phi(L|Tp)$, and two first-order factors modeling the correlation of the temperature readings between the 1^{st} and 2^{nd}, and 3^{rd} and 4^{th} sensor pairs in every room, $\phi(Tp_2|Tp_1)$ and $\phi(Tp_4|Tp_3)$.

We used selection queries over the *Sensor* table with different selectivity. We also generated sensor readings with different sizes and different percentage of missing values. Next we describe our data generator.

Data Generation

We generated a synthetic dataset, by forward sampling on the model described earlier. In order to thoroughly evaluate the effectiveness of our probabilistic selection algorithms, we varied a number of parameters.

- Data Size (*size*): We generated the *Sensor* table with different number of tuples (i.e., sensor readings) by increasing the number of timestamps;
- Data Missing Ratio (*mratio*): We generated the *Sensor* table with different percentage of missing values in order to vary the level of uncertainty in the data;

- Query Selectivity Ratio (*sel*): We generated selection queries with different selectivity by appropriately manipulating the CPTs of the first order factors $\phi(Tp_2|Tp_1)$ and $\phi(Tp_4|Tp_3)$.
- Model Connectivity Ratio (*cratio*): The model connectivity '*cratio*' is the number of *vertical correlations* between tuples over the total number of tuples in *Sensor*. Connectivity ratio was varied by adding or dropping first-order factors in the model described in Section 4.5.1.

Based on these parameters, we generate data sets, queries, and models that has different characteristics, which influence the size of the output and the runtime of the probabilistic selection algorithms.

Algorithms and Metrics

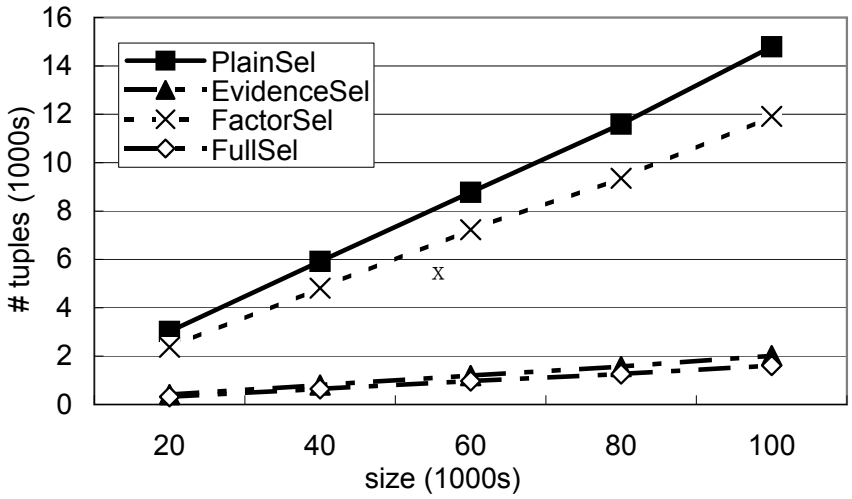
We evaluate the four probabilistic selection algorithms over incomplete relations with different optimizations presented in Section 4.2.2. Algorithm `PlainSel` filters data tuples of an incomplete relation R using the traditional selection operator, and generates evidence tuples, resorting to a full transitive closure over R and the model structure. `EvidenceSel` uses the "evidence-based early-stopping" technique on top of `PlainSel`, while `FactorSel` extends `PlainSel` by incorporating the "factor-based data-filtering" technique. Lastly, `FullSel` uses both optimizations.

4.5.2 Data Size

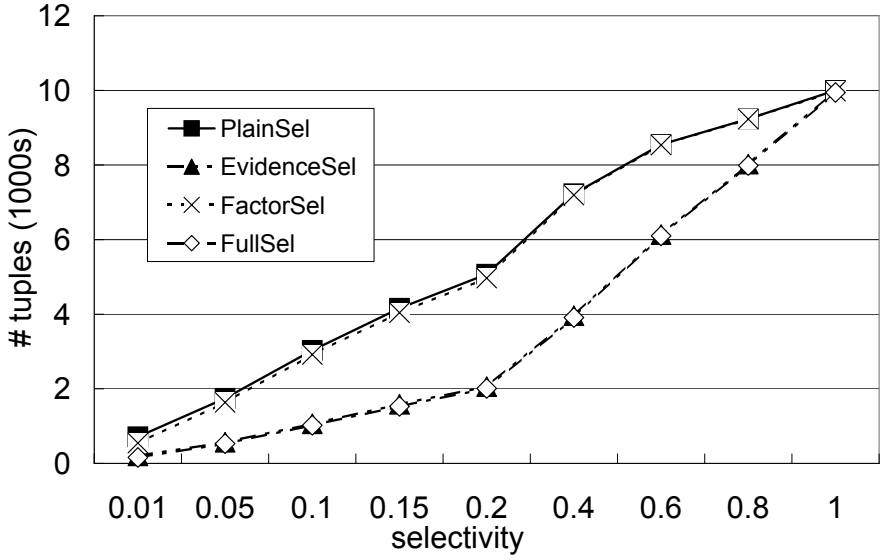
In this first experiment, we varied the number of tuples (i.e., sensor readings) in the *Sensor* table from $20,000$ to $100,000$, while keeping the query selectivity (0.01), the model connectivity ($7/8$), and the percentage of missing values (0.01) constant. The "evidence-based early-stopping" technique employed in `FullSel` and `EvidenceSel` (behind the `FullSel` line in Figure 4.8(a)) reduces the size of the result relation significantly compared to `PlainSel` and `FactorSel`. By comparing `FactorSel` and `PlainSel`, we observe that the "factor-based data-filtering" technique employed by the former also decreases the result size, but not as much as the "evidence-based early-stopping" technique.

As shown in Table 4.10, the runtime of all probabilistic selection algorithms for data with size $10,000$ is below 0.5 second. The difference in runtime among these algorithms is within a factor of 2. The subsequent inference algorithm, on the other hand, is 20 to 100 times more expensive than the probabilistic selection algorithms in runtime. Thus, the comparative execution time of various probabilistic selection algorithms is not the focus of this evaluation.

The runtime of the subsequent inference algorithm, as we can see in Table 4.10, is proportional to the size of the input (i.e., the output of the probabilistic selection algorithm). For example, while the output size of the `PlainSel` is 10 times that of the `FullSel`, the inference runtime over the output of the `PlainSel` algorithm is 5 times that of the `FullSel` algorithm. Thus, the focus of the experiments in this Chapter is to compare the size of the output among different probabilistic selection algorithms.

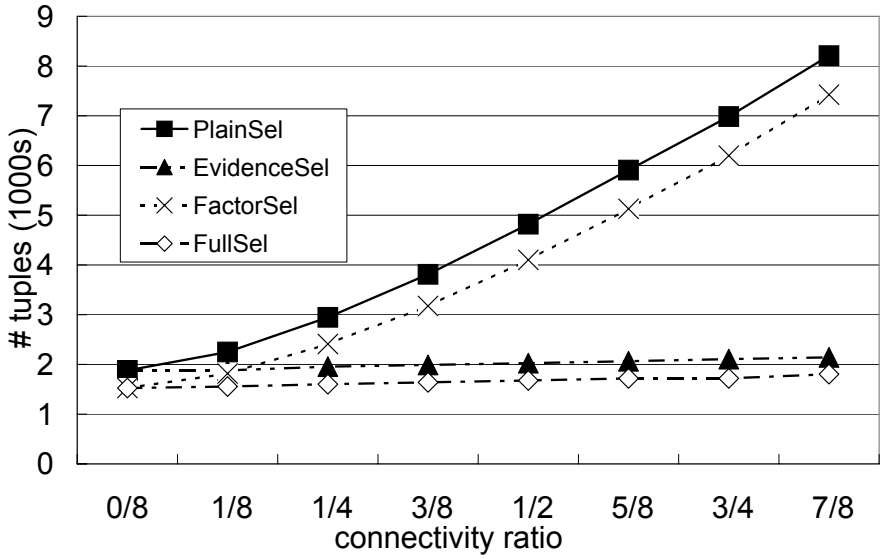


(a) Result size, varying R's size, while keeping $\{sel = 0.01$, $cratio = 7/8$, $mratio = 0.01\}$ constant.

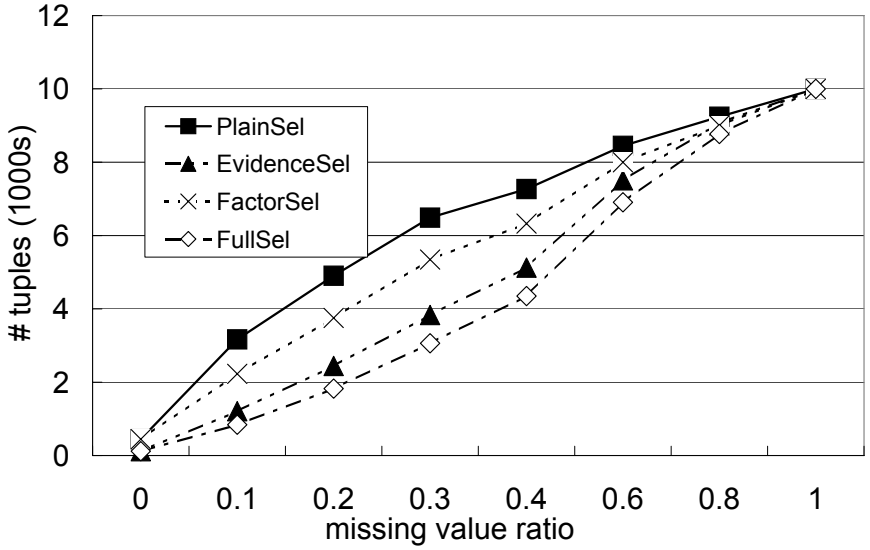


(b) Result size, varying query sel, while keeping $\{size = 10000$, $cratio = 1/2$, $mratio = 0.01\}$ constant.

Figure 4.8. Output size of various probabilistic selection algorithms for different sizes of the incomplete relation SENSOR and different selectivity of the queries.



(a) Result size, varying the model's connectivity ratio, while keeping $\{$*size* $=$ 10000, *sel* $= 0.1$, *mratio* $= 0.1\}$ constant.



(b) Result size, varying the missing values' ratio, while keeping $\{$*size* $= 10000$, *sel* $= 0.01$, *cratio* $= 1/2\}$ constant.

Figure 4.9. Output size of various probabilistic selection algorithms for different percentages of missing values in SENSOR and different connectivity ratios of the model.

	PlainSel	FactorSel	EvidenceSel	FullSel
Probabilistic Selection Runtime (sec)	0.28	0.32	0.20	0.33
Input Size (number of tuples)	1512	1184	204	159
Regular Inference Runtime (sec)	27.1	21.9	5.73	5.53

Figure 4.10. Probabilistic selection runtime vs. size of output (i.e., input to inference) vs. inference execution time for a selection query with *sel* $= 0.01$, data with *size* $= 10000$, *mratio* $= 0.01$, and a model with *cratio* $= 7/8$.

4.5.3 Data Uncertainty

In the second experiment, we varied the selectivity *sel* of the queries, while keeping its size to $10,000$ tuples, connectivity ratios as 0.5, and the percentage of missing values as 0.01. As shown in Figure 4.8(b), `PlainSel` and `FactorSel` generate similar output sizes, while `FullSel` and `EvidenceSel` generate similar output sizes. In addition, we see that `PlainSel` generates more than twice as many tuples as the `FullSel` algorithm when the selectivity is below 0.2. As the selectivity increases, the difference in the size of the output among different algorithms decreases, until it becomes zero when the selectivity is 1. This is because the "evidence-based early-stopping" technique can no longer filter out a lot of tuples when all the tuples are selected. Also, in this setup, very litter improvement is achieved by using "factor-based data-filtering" technique.

4.5.4 Model's Connectivity Ratio

The third parameter we varied was the model's connectivity ratio, keeping this time the size of the *Sensor* table $(10,000)$, the percentage of missing value (0.1), and the query's selectivity (0.1) constant. Figure 4.9(a) clearly shows that, as the model connectivity increases, the sizes of the output of different probabilistic selection algorithms increase linearly as well. However, the output sizes of the algorithms `PlainSel` and `FactorSel` grow much faster than output sizes of the algorithms `EvidenceSel` and `FullSel`, because they do not utilize the "evidence-based early-stopping" technique. Algorithms `EvidenceSel` and `FullSel`, on the other hand, only have minimal increase in the result size of around 10%.

4.5.5 Data Uncertainty

We also experimented by varying the ratio of missing values '*mratio*' in the *Sensor* table, while keeping its size to $10,000$ tuples, query's selectivity to 0.01, and model connectivity ratio to 0.5. As we can see in Figure 4.9(b), `EvidenceSel` and `FullSel` perform better than `PlainSel` and `FactorSel` in all cases, except for when the percentage of the missing value is 1, in which case, all tuples are retrieved by all the algorithms.

From the above four experiments, we show that probabilistic selection algorithms `EvidenceSel` and `FullSel` that utilizes the "evidence-based early-stopping" technique always perform better than their counterparts: `PlainSel` and `FactorSel` algorithms respectively. While

Runtime (sec)	PlainSel	FactorSel	EvidenceSel	FullSel
Regular Inference	27.1	21.9	5.73	5.53
First-order Inference (Best-Case)	2.0	1.6	1.1	0.9

Figure 4.11. Inference execution time for selection query with *sel* $= 0.01$, data with *size* $= 10000$, *mratio* $= 0.01$, and model with *cratio* $= 7/8$.

the "factor-based data-filtering" technique does reduce the size of the output as well, it is not as effective as the "evidence-based early-stopping" technique in our experimental setup.

4.5.6 First-order Inference

One of BAYESSTORE's major objectives is the integration of relational queries with probabilistic inference operators. In our final experiment, we computes the joint distribution F of the RVs that correspond to the missing values in the *Sensor* table, for all the readings that satisfy the condition `L='Drk'`. According to our query semantics, this inference query produces a new incomplete relation *Sensor*' and model $\mathcal{M}_{FOBN}$'.

The selection query operates both on the model $\mathcal{M}_{FOBN}$, using the `Select-Model` algorithm from Section 4.2.1, and on *Sensor*, using different data selection algorithms from Section 4.2.2. The resulting model $\mathcal{M}_{FOBN}$' is grounded to a flat Bayesian Network, so that standard inference algorithms can be applied. Due to time limitations, rather than integrating an inference operator in our prototype, we used the implementation of the Variable Elimination algorithm from Kevin Murphy's BNT toolbox. On top of it, we implemented one of the First-Order inference techniques discussed in [58] (the `FirstorderSharing` technique), which allows us to share the inference computation over a population of instances with the same model template, without fully grounding the model. Thus, if every sensor in a sensor deployment has an independent and identical probabilistic model, then the inference computation over every sensor reading is the same. Thus, the inference computation can be shared among all sensor readings.

Table 4.11 shows the total execution time for the query, using our various data selection algorithms, with and without the first-order optimizations. This is a best-case scenario, because the probabilistic model over sensors in each room is the same in our model. Thus, we can share the inference computation among different rooms over different timestamps. In this case, the first-order inference optimization results in a 5 to 10 times reduction on the query execution time. We also notice that with the first-order inference optimization, the `FullSel` algorithm can still achieve a 2 times speedup compare to the `PlainSel` algorithm.

This best-case evaluation show the promising potential to speed-up inference algorithms using first-order structures in FOBN models. However, first-order inference does not always reduce inference runtime. In fact, there are cases where the overhead can outweigh the improvement of first-order inference in runtime. An optimizer is needed to be able to choose between different inference algorithms and optimizations. In Chapter 7, we show our effort in that direction.

These results indicate that data and evidence filtering techniques manage to reduce the time

of the inference operator, as a welcome effect of the incomplete relation's size reduction. The increased cost for the select operation is more than ten times less than the reduced cost for the inference operation.

4.6 Summary

In this chapter, we described the new algorithms to execute basic relational operators over an instance of the BAYESSTORE data model for the application of sensor networks based on FOBN models. Each relational operator, including select, project, and join, consists of a two-part computation: one part over the incomplete relation and the second part over the FOBN model. We evaluated different variations of these algorithms in terms of the size of the output. We showed that the effectiveness of pruning the output tuples directly affects the efficiency of the follow-up inference algorithms. Furthermore, a preliminary experiment also showed the benefits of utilizing first-order structures in the FOBN models in improving the runtime of the inference algorithms. In this chapter, we focused on the FOBN model for sensor network applications. We touched upon the inference runtime related to evaluating the efficiency of the probabilistic relational operators. In the next chapter, we focus on the implementation and evaluation of inference algorithms for information extraction over text.

Chapter 5

Inference Operators

In this chapter, we describe the in-database implementation of various inference algorithms over probabilistic graphical models, including Viterbi, sum-product, and MCMC algorithms described in Chapter 2.2. In this thesis, we built two instances of the BAYESSTORE system using the same data model and system architecture described in Chapter 3. For the rest of this thesis, we describe the development of the BAYESSTORE system components for inference, probabilistic query processing, and optimization driven by IE applications over text data. The inference algorithms are described over this BAYESSTORE instance with the linear-chain CRF model.

We first describe the schema in this BAYESSTORE instance to store the CRF model. Then we describe the detailed implementation of the inference algorithms over CRF models. Some of the inference algorithms, such as Viterbi, are restricted to linear-chain CRF models; while other inference algorithms, such as the MCMC Gibbs sampler, are generally applicable to all PGMs, including CRF and FOBNs. Finally, we show the runtime evaluation of the in-database implementation of these inference algorithms compared to the corresponding open-source implementations. The results show that inference algorithms over probabilistic graphical models can be efficiently implemented in a set-oriented programming and execution framework, such as a relational database.

5.1 Overview

Our in-database implementation of the inference algorithms over the CRF model for IE is based on the following two observations:

- *Relational Representation of Inputs:* As shown in Section 3.4, CRFs can be presented as a first-class object in a BAYESSTORE. Similarly, text data can be captured relationally via the inverted file representation commonly used in information retrieval.
- *Declarative Inference:* Given tabular representations of CRF model parameters and input text, the inference algorithms for CRFs can be elegantly expressed as SQL queries using

token	prevLabel	label	score
2181 (DIGIT)	null	street num	22
2181 (DIGIT)	null	street name	5
...	..	..	
Berkeley	street	street name	10
Berkeley	street	city	25
..	..	..	

Figure 5.1. An instance of the MR table.

advanced features in PostgreSQL 8.4, such as recursive joins, window functions and array data types.

Together, these observations result in a unified and efficient approach for implementing CRF and its inference algorithms using the PostgreSQL DBMS. In Section 5.4.2, we show that the runtime performance of this system is comparable to the corresponding standalone open-source implementations of those inference algorithms. Our approach not only correctly performs CRF-based IE on input text, but also maintains the probability distributions inherent in CRF models to support probabilistic analysis queries over the uncertain extraction results.

5.2 MR Matrix: A Materialization of the CRF Model

Before we delve into the detailed implementation of the various inference algorithms over the CRF model, let us first look at the relation and the corresponding schema that we use to represent the CRF model. This model encodes the probability distribution F over all possible extractions. The CRF model is stored in the MR matrix (the name MR matrix is borrowed from [66]), which is a materialization of the factor tables in the CRF model for all the tokens in a document $\mathcal{D}$. More specifically, each token x_t in $\mathcal{D}$ is associated with a factor table $\phi[y_t, y_{t-1} \mid x_t]$ in the CRF model, which represents the correlations between the token x_t, the label y_t and the previous label y_{t-1}. The factor table $\phi[y_t, y_{t-1} \mid x_t]$ is computed from the weighted sum of the features activated by x_t in the CRF model:

$$\phi[y_t, y_{t-1} \mid x_t] = \sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t).$$

where the features are real-valued functions, as described in Section 3.4. There are two ways to store the MR matrix. The first is to use the following schema, where {token,label,prevLabel} is the primary key:

MR (token, label, prevLabel, score)

An example of an MR matrix instance of the above schema is shown in Fig. 5.1. The second way is to store the factor table $\phi[y_t, y_{t-1} \mid x_t]$ for each token x_t as an array data type, where the array contains a set of scores sorted by {prevLabel, label}. This is a more compact representation, and can lead to better memory locality characteristics. In addition, with the array data type, we do not have to explicitly store the values of prevLabel and label, we can simply look up the score by index. For example, if we want to fetch the score for prevLabel=5 and label=3, then we look up the $(5 \times |Y| + 3)$th cell in the array. The MR matrix schema with the array data type is:

MR (token, score ARRAY[])

The TOKENTBL and the MR table are the relational and the probabilistic components of the BAYESSTORE instance for information extraction over text. TOKENTBL stores text in a relational form, and MR stores the CRF model in a relational form. The probabilistic distribution of the missing label^p values in TOKENTBL is encoded in the MR table. Together, these two tables represent the uncertainties and probability distribution needed for probabilistic IE and query processing over its probabilistic outcome.

5.3 Top-k Inference over CRF

This section describes the recursive SQL implementation of the Viterbi algorithm – the central inference algorithm to compute the top-k extractions of structure data over unstructured text using linear-chain CRF models. We compare the merits of a declarative SQL implementation vs. an imperative Java implementation of the Viterbi algorithm, and decide on a middle ground that retains a good deal of the declarativeness of SQL, but embeds imperative UDF functions for issues where relational is not a good representation, such as vectors and arrays. We also implemented the sum-product algorithm over linear-chain CRF model in a similar manner.

5.3.1 Viterbi SQL Implementations

There are three approaches to implement the Viterbi algorithm: declarative, imperative, and hybrid. The Java implementation of the Viterbi algorithm in the CRF open source project [66] is fully imperative. The Viterbi dynamic programming algorithm can also be implemented declaratively using recursive SQL queries over the incomplete relation TOKENTBL and the model representation in the MR matrix. In this section, we describe ViterbiPerStr and ViterbiAllStr, two fully declarative SQL implementation of the Viterbi algorithm.

The declarative and imperative approaches each have their own pros and cons. The declarative approach makes the Viterbi algorithm more concise and easier to write, and opens up cost-based optimization opportunities. On the other hand, the imperative approach is more flexible in applying customized optimizations, which may lead to a large performance advantage over the declarative approach.

Instead, we choose a middle ground that retains a good deal of the declarativeness, but embeds imperative UDFs for issues where relational is not a good representation, such as vectors and ordered lists. This SQL implementation with special UDF functions is called ViterbiArray.

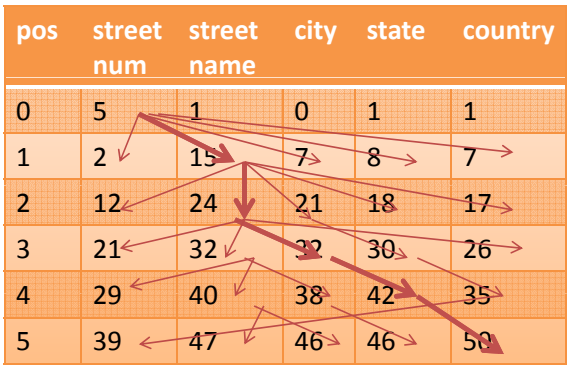


Figure 5.2. Illustration of computing V matrix in ViterbiPerStr algorithm.

ViterbiPerStr and ViterbiAllStr: As stated in Section 2.2.1, the Viterbi algorithm computes the top-k segmentation using a dynamic programming data structure – the V matrix. Let us assume that we are computing top-1 segmentation for simplicity. Each cell in $V(i, y)$ stores the score and the path of the top-1 partial segmentation up until position i ending with label y. The V matrix at position 0 is initialized from the MR matrix: $V(0, y) = \phi[y, -1|x_0]$, where -1 denotes that the previous label is `NULL`. The V matrix at position $i > 0$ is recursively defined from the V matrix at position $i - 1$, by picking the partial segmentation with maximum score: $V(i, y) = \max_{y'}\{V(i - 1, y') + \phi[y, y'|x_i]\}$. The next example illustrates the score and the path of the top-1 segmentations stored in the V matrix.

Example 9 *We use the address of "Jupiter", a popular jazz bar in downtown Berkeley, "2181 Shattuck Ave. Berkeley CA USA" as an example. Fig 5.2 shows the V matrix computed by the ViterbiPerStr algorithm. The first row contains the possible labels and the first column contains the string positions from 0 to 5. The scores are shown in the cells of the V matrix; the path of the top-1 partial segmentations are shown as the edges. For example, the partial segmentation in V(3,'city') consists of three edges V(0,'street num') → V(1,'street name'), V(1,'street name') → V(2,'street name'), and V(2,'street name') → V(3,'city').*

The top-1 segmentation of the text-string can be computed by following the path from the cell in the V matrix with the maximum score. In this example the path is high-lighted: {V(0,'street num') → V(1,'street name') → V(2,'street name') → V(3,'city') → V(4,'state') → V(5,'country')}. □

The basic SQL implementation of any dynamic programming algorithm involves recursion and window aggregation. In the Viterbi algorithm, the window aggregation is `group by` followed by `sort` and `top-1`. In Fig. 5.3, we show the SQL implementation for the ViterbiPerStr algorithm

```
1 CREATE FUNCTION ViterbiPerStr (int) RETURN VOID AS
2 $$
3 -- compute the top-1 path from V
4 INSERT INTO Ptop1
5 WITH RECURSIVE P(pos,segID,label,prevLabel,score) AS (
6     SELECT * FROM Vtop1 ORDER BY score DESC LIMIT 1
7     UNION ALL
8     SELECT V.* FROM Vtop1 V, P
9     WHERE V.pos = P.pos-1 AND V.label = P.prevLabel
10 ),
11 -- compute the V matrix from mr and tokenTbl
12 Vtop1 AS(
13   WITH RECURSIVE V(pos,segID,label,prevLabel,score) AS (
14     -- compute V(0,y) from mr and tokenTbl
15     SELECT st.pos, st.segID, mr.label, mr.prevLabel, mr.score
16     FROM tokenTbl st, mr
17     WHERE st.strID=$1 AND st.pos=0 AND mr.segID=st.segID
18          AND mr.prevLabel=-1
19     UNION ALL
20     -- compute V(i,y) from V(i-1,y), mr and tokenTbl
21     SELECT start_pos, seg_id, label, prev_label, score
22     FROM (
23       SELECT pos, segID, label, prevLabel, score, RANK()
24            OVER (PARTITION BY pos,label ORDER BY score DESC) AS r
25       FROM (
26           SELECT st.pos, st.segID, mr.label, mr.prev_label,
27                 (mr.score+v.score) as score
28           FROM tokenTbl st, V, mr
29           WHERE st.strID=$1 AND st.pos = v.pos+1
30                AND mr.segID=st.segID AND mr.prevLabel=v.label
31       ) as A
32     ) as B WHERE r=1
33 )SELECT * FROM V)
34 SELECT $1 as strID, pos, segID, label FROM Ptop1
35 $$
36 LANGUAGE SQL;
```

Figure 5.3. ViterbiPerStr UDF function in SQL takes in one strID at a time and computes the top-1 segmentation.

in a UDF function in SQL. ViterbiPerStr processes one strID at a time, whereas an alternative is to compute the top-k inference for all text-strings in $\mathcal{D}$ at the same time, which we call the ViterbiAllStr algorithm. However, ViterbiAllStr suffers from generating a large intermediate table that cannot be indexed.

Lines 11 – 33 compute the V matrix in the Viterbi algorithm. Lines 14 – 18 initialize the V matrix at position 0 for all labels. Lines 26 – 30 compute all possible partial segmentations up until position i by joining the portion of the V matrix at position $i-1$: $V(i-1)$ and the portion of MR table for the token at position i: $MR(x_i)$ on condition $V(i-1).label = MR(x_i).prevLabel$. The rest of the logic in lines 20 – 33 computes the top-1 partial segmentation $V(i, y_i)$ among all the partial segmentations up until position i with the same y_i value, using the `rank()` (for computing top-1) function with `partition by` (group by) and `order by` (sort).

After the V matrix is computed, lines 3 – 10 reconstruct the top-1 segmentation for a text-string, by backtracking from the maximum score in the V matrix. Line 6 picks the cell in the V matrix with the maximum score, and lines 8,9 recursively trace back the top-1 segmentation.

The mode of computation for ViterbiPerStr is to execute the inference for one text-string at a time. The most important change for ViterbiAllStr from ViterbiPerStr is omitting the strID$= \$1$ predicate on Lines 17 and 29. In this case, the V matrix becomes three dimensional with strID, position and label.

ViterbiArray: The ViterbiArray algorithm is an optimization of the ViterbiPerStr algorithm, which uses the second schema of the MR matrix from Section 5.2, and takes advantage of the array data type. Correspondingly, the `score` in the V matrix for a specific position i is also stored as an array of $\langle score, prevLabel \rangle$ pairs with length $|Y|$, where prevLabel is used for backtracking the top-1 segmentation.

Fig. 5.4 shows the SQL statements in ViterbiArray that replace Line 21 – 32 of ViterbiPerStr, where the UDF function TOP1-ARRAY$(V(i-1), MR(x_i))$ is used to replace the join between array $V(i-1)$ and array $MR(x_i)$, and the window aggregation (group-by, sort and top-1 operations) over the result of the join.

Figure 5.4 also shows the algorithm for the function TOP1-ARRAY, which takes in two arrays: the first is two dimensional `arrayV` $V(i-1)$ and the second is one dimensional `arrayMR` $MR(x_i)$. `arrayV` is a 2 by $|Y|$ array, and `arrayMR` is a $|Y| \times |Y|$ array. Lines 10 – 11, join `arrayV` and `arrayMR` and compute the new scores for the join result tuples. Line 13 – 17 compute the top-1 $\langle score, prevLabel \rangle$ pair for each label y_i. The function returns a new `arrayV` $V(i)$.

The ViterbiArray algorithm is much faster than the ViterbiPerStr algorithm, because (1) the join on $V(i-1).label = MR(x_i).prevLabel$ is replaced by index computation between two arrays $V(i-1).score$ and $MR(x_i).score$; (2) the scores in the MR matrix are compactly represented as an array, which greatly improves the memory locality characteristics; (3) the computation of the group by, sort and top-1 computation in ViterbiPerStr is replaced by index computation and the maintenance of a priority queue.

```
1 SELECT st.pos, st.segID,
2       top1_array(v.score, mr.score) as score
3 FROM tokenTbl st, V, mr
4 WHERE st.strID=$1 AND st.pos = v.pos+1
5       AND mr.segID=st.segID
```

TOP1-ARRAY $(int[2][]arrayV, int[]arrayMR)$

1 $size1 =$ *length of dimension 2 of arrayV*;

2 $size2 =$ *length of dimension 1 of arrayMR*;

3 $result =$ *new* $int[2][size1]$;

4

5 *// compute the top1 for each label over the join result*

6 *// of arrayV and arrayMR on V.label=MR.prevLabel*

7 *// arrayMR is ordered by (prevLabel,label)*

8 **for** $i = 0; i < size2; i$++ **do**

9 *// prevLabel and label in arrayMR*

10 $k = i/size1; k_r = i\%size1;$

11 $newscore = arrayV[0][k] + arrayMR[i];$

12

13 **if** $k == 0 \parallel newscore > result[0][k_r]$ **then**

14 $result[0][k_r] = newscore;$

15 *// record the prevLabel for the top1 for label* k_r

16 $result[1][k_r] = k;$

17 **endif endfor**

18 **return** $result$

Figure 5.4. ViterbiArray modification in ViterbiPerStr and the algorithm for UDF function in C top1-array.

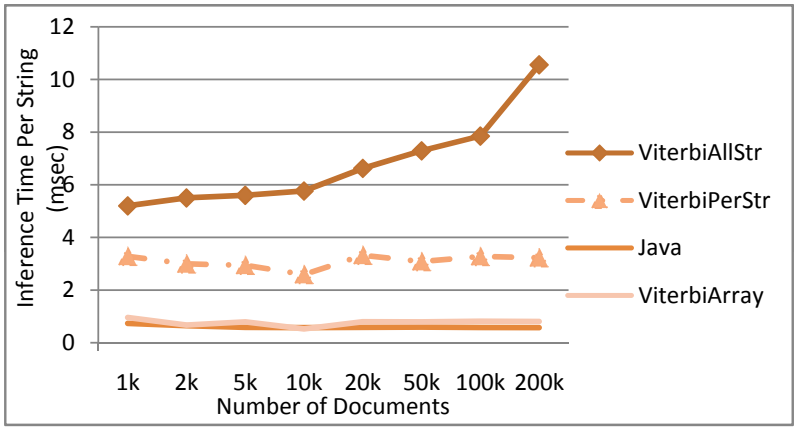


Figure 5.5. Average inference time (msec) for a single text-string for different implementations of the Viterbi algorithm.

dataset	ViterbiAllStr	ViterbiPerStr	ViterbiArray	Java
address	10.5 msec	3.2 msec	0.8 msec	0.5 msec
bib	1760.1 msec	175.1 msec	6.2 msec	16.2 msec

Figure 5.6. Average inference time per text-string (msec) for different Viterbi implementations on address and bib dataset.

5.3.2 Experimental Results

We implemented the Viterbi inference algorithm over CRF models in the PostgreSQL 8.4 development version. The reason we use the development version is that it supports recursive queries. The Java implementation of the CRF model learning and inference is from the CRF open source project [66]. We conducted our experiments on a 2.4 GHz Intel Pentium 4 Linux system with 1GB RAM.

We use two datasets in the evaluation. The first dataset is a set of over $200k$ address strings we extracted from the yellow pages. The average number of tokens in the address strings is 6.8, and 8 labels are used to tag this dataset. The second dataset is a set of more than $18k$ bibliography entries prepared by R.H. Thomason [67]. The average number of tokens in the bibliography strings is 37.8, and 27 labels are used to tag this dataset. We ran the three top-k inference implementations in SQL: ViterbiAllStr, ViterbiPerStr and ViterbiArray, and the Java implementation on the address dataset.

Fig. 5.5 shows the average inference time per text-string (IPS) in msec for different Viterbi implementations with increasing number of text-strings on the x-axis, from $1k$ to $200k$. The results show that ViterbiAllStr is not scalable with the number of text-strings, and ViterbiPerStr is 6 times more expensive than the hand-tuned Java implementation. On the other hand, the use of the array data type in ViterbiArray demonstrates comparable performance to the Java implementation. The average IPS is 0.57 msec for Java, and 0.81 for ViterbiArray.

Fig. 5.6 compares the IPS numbers over two datasets. The ViterbiArray implementation is more efficient than the Java implementation on the bibliography dataset. There are two main reasons for this: (1) ViterbiArray uses the materialized MR matrix, which needs to be computed on the fly in the Java implementation; and (2) ViterbiArray uses an efficient join between arrays with good memory locality characteristics, which is especially evident with a large number of labels.

5.4 MCMC Algorithms

In this section, we describe the in-database implementation of a different class of inference algorithms—the MCMC inference algorithms, which are much more general than the Viterbi algorithm. The MCMC algorithms can be applied to perform inference not only over models with a special restricted structure, such as linear-chain CRF, but can also over general probabilistic graphical models, such as Bayesian Networks and Markov Random Fields. We describe the implementation of two MCMC algorithms: the Gibbs sampler and Metropolis Hastings.

5.4.1 SQL Implementation of MCMC Algorithms

Both the Gibbs sampler and the Metropolis Hastings (MCMC-MH) algorithm are iterative algorithms that contain three main steps: 1) initialization, 2) generating proposals, and 3) generating samples. They differ in their proposal and sample generation functions.

```
1  CREATE FUNCTION Gibbs (int) RETURN VOID AS
2  $$
3  -- compute the initial world: genInitWorld()
4  insert into MHSamples
5  select setval('world_id',1) as worldId, docId, pos, token,
           trunc(random()*num_label+1) as label
6  from tokentbl;

7  -- generate N sample proposals: genProposals()
8  insert into Proposals
   with X as (
     select foo.id, foo.docID, (tmp\%bar.doc_len) as pos
9    from (
         select id, ((id-1)/($1/numDoc)+1) as docID,
           ((id-1)\%($1/numDoc)) as tmp
10       from generate_series(1,$1) id) foo, doc_id_tbl bar
11       where foo.doc_id = bar.doc_id
     )
12 select X.id,S.docId,S.pos,S.token, null as label,
     null::integer[] as prevWorld, null::integer[] as factors
13 from X, tokentbl S
14 where X.docID = S.docID and X.pos = S.pos;

15 -- fetch context: initial world and factor tables
16 update proposals S1
17   set prev_world = (select * from getInitialWorld(S1.docId))
18   from proposals S2
19   where S1.docId <> S2.docId and S1.id = S2.id+1;
20 update proposals S1
21   set factors = (select * from getFactors(S1.docId))
22   from proposals S2
23   where S1.docId <> S2.docId and S1.id = S2.id+1;

24 -- generate samples: genSamples()
25 insert into MHSamples
26 select worldId, docId, pos, token, label
27 from (
28     select nextval('world_id') worldId, docId, pos, token, label,
                getalpha_agg((docId,pos,label,prev_world,factors)
                ::getalpha_io) over (order by id) alpha
29     from (select * from proposals order by id) foo) foo;
30 $$
31 LANGUAGE SQL;
```

Figure 5.7. The SQL implementation of Gibbs sampler takes input N – the number of samples to generate.

We initially implemented the MCMC algorithms in the SQL procedure language provided by PostgreSQL—PL/pgSQL—using iterations and three User Defined Functions (UDF's):

- GENINITWORLD() to compute the initialized world (line 1 for Gibbs in Figure 5.7);
- GENPROPOSAL() to generate one sample proposal (line 3 for Gibbs in Figure 5.7);
- GENSAMPLE() to compute the corresponding sample for a given proposal (line 4 for Gibbs in Figure 5.7).

However, this implementation ran hundreds of times slower than the Scala/Java implementation described in [68]. This is mainly because calling UDF's iteratively a million times in a PL/pgSQL function is similar to running a SQL query a million times. A more efficient way is to "decorrelate", and run a single query over a million tuples. The database execution path is optimized for this approach. With this basic intuition, we re-implemented the MCMC algorithms, where the iterative procedures are translated into set operations in SQL.

The efficient implementation of the Gibbs sampler is shown in Figure 5.7, which uses the feature of *window functions* introduced in PostgreSQL 8.4. MCMC-MH can be implemented efficiently in a similar way with some simple adaptations. The pseudo-code of MCMC-MH can be found in Appendix A.1.

Lines 3 to 6 of Figure 5.7 show one way to generate the initial world, by randomly assigning *labels* [1] to each token in TOKENTBL. Lines 7 to 14 show how to generate N proposals of label variables to sample next, using a single SQL query rather than iteratively (N being represented as the parameter $1). Relation `X` is the "skeleton" of the proposals, consisting of the `id` of the sample, the `docID` and the `pos` of the token in TOKENTBL to perform the next sample. The "skeleton" `X` is then joined with TOKENTBL to generate a set of proposals. For MCMC-MH, labels of the proposed variables are also randomly assigned.

Lines 15 to 23 update the PROPOSALS table by initializing the first proposal of each document with its initial world and the corresponding factor tables. This is the "context" for the first sample proposal of each document; for each following sample proposal, the "context" is passed as the aggregate states of the window function. As shown in lines 24 to 30, the sample corresponding to each proposal is computed using the window aggregate function `getalpha_agg`. `getalpha_agg` takes in the current proposal `(docID, pos)`, and the "context" `(prev_world, factor)`, and returns the new sample `(docID,pos,label)` based on the proposal distribution. For MCMC-MH, `getalpha_agg` computes α—the acceptance ratio of the proposal—and either accepts the proposal by returning it as the sample `(docID,pos,label)`, or rejects it by returning `NULL`.

This implementation achieves similar (within a factor of 1.5) runtime compared to the Scala/Java implementation of the MCMC algorithms, as shown in the next section.

5.4.2 Experimental Results

We now present the results of our experiments to evaluate the efficiency of the SQL implementation of the MCMC methods. We implemented the MCMC inference algorithms in PostgreSQL 8.4.1.

[1] numLabel is the total number of possible labels.

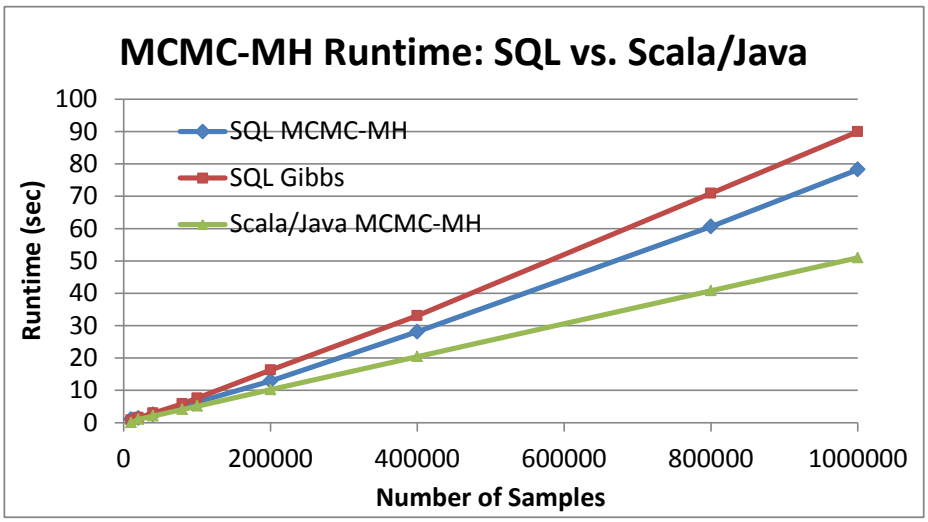


Figure 5.8. Runtime comparison of the SQL and Java/Scala implementations of MCMC-MH and Gibbs algorithms over DBLP.

We conducted the experiments reported here on a 2.4 GHz Intel Pentium 4 Linux system with 1GB RAM.

For evaluating the efficiency of the in-database implementation of the MCMC methods, we use the DBLP dataset [69] and a CRF model with 10 labels and features similar to those in [20]. The DBLP database contains more than $700,000$ papers with attributes, such as conference, year, etc. We generate bibliography strings from DBLP by concatenating all the attribute values of each paper record. We also have similar results for the same experiments over NYTimes dataset.

In the following experiment, we compare the runtime of the in-database implementation of the MCMC algorithms, including Gibbs Sampling and MCMC-MH, with the runtime of the Scala/Java (with Scala 2.7.7 and Java 1.5) implementation of MCMC-MH described in [68] over linear-chain CRF models. The runtime of Scala/Java implementation is measured on a different machine with better configurations (2.66GHz CPU Mac OSX 10.6.4 with $8G$ RAM).

As shown in Figure 5.8, the runtimes of the MCMC algorithms grow linearly with the number of samples for both the SQL and the Java/Scala implementations. While the Scala/Java implementation of MCMC-MH can generate 1 million samples in approximately 51 seconds, it takes about 78 seconds for the SQL implementation of the MCMC-MH, and about 89 seconds for that of the Gibbs Sampling. The result shows that, in the setup of this experiment, the in-database implementations of the MCMC sampling algorithms achieve comparable (within a factor of 1.5) runtimes to the Java/Scala implementation.

5.5 Guidelines for Implementing In-database Statistical Methods

Having implemented a number of inference algorithms in the database, including the Viterbi dynamic programming for linear-chain CRF, sum-product algorithms (refer to Section 7), and the

MCMC methods described above, we have developed a set of design guidelines for implementing statistical methods in the database:

- *Avoid using iterative programming patterns in PL/pgSQL and other database extension languages*. This is easily overlooked, since many machine learning inference techniques are described as iterative methods. Database architectures are optimized for running a single query over lots of data, rather than iteratively running a little query over small amounts of data. The passing of "iterative state" is achieved in a single query expression via recursive queries in Viterbi and sum-product, and window aggregate functions in MCMC-MH and Gibbs.

- *Use efficient representations for factor tables.* Tables and rows are heavy-weight representations for cells in a factor table. Array data types (which are now standard in many relational database systems) provide better memory locality, faster look-up and more efficient operations.

- *Drive the data-flow via a few SQL queries, and use user-defined functions (and aggregates) to do inner-loop arithmetic.* This design style enables the database optimizer to choose an efficient data-flow, while preserving programmer control over fine-grained efficiency issues.

- *Keep running state in memory, and update using user-defined functions and aggregates.* This is typically far more efficient than storing algorithm state in the database and updating using SQL.

5.6 Summary

In this chapter, we described the in-database implementation of two classes of inference algorithms over the CRF model. The first class contains exact inference algorithms that computes top-k extractions over the linear-chain CRF model. These algorithms cannot be applied to more general probabilistic graphical models. The second class contains general but approximate inference algorithms—the MCMC algorithms. This class of inference algorithms can be applied over graphical models with more complex structures. The experimental results show that these statistical methods can be implemented efficiently inside of a relational database, following a set of implementation rules. The native implementation of the inference algorithms opens up opportunities for deep integration and optimization between the inference and the relational operators in the database. We show such optimizations in the next chapter.

Chapter 6

Viterbi-based Probabilistic Query Processing

Having described the efficient in-database implementation of the inference algorithms over CRF models in BAYESSTORE, we can now execute probabilistic analysis queries that involve both relational and inference operators over the extracted entities from unstructured text. In this chapter, we focus on the set of probabilistic queries that can be answered using the top-k inference algorithm over the linear-chain CRF models—Viterbi. We evaluate such query processing and describe new algorithms for probabilistic queries, which optimize by reordering the operators in query execution and by employing a deep integration between the relational and inference operations. We conduct experiments comparing both the runtime efficiency and answer quality of these new algorithms with the standard query execution plans.

6.1 Overview

In Chapter 3 and Chapter 5, we described an in-database implementation of the CRF model and its maximum-likelihood (ML[1]) inference algorithm—Viterbi. As we described in Section 1.4.3, taking only the ML extraction results and ignoring their associated probabilities enables relational queries to be performed in a straightforward manner. This approach, however, suffers from two important limitations: (1) as shown in [28], restricting the queries to the ML extractions can result in incorrect answers, and (2) because the IE inference is computed separately from the queries over the extracted data, important optimization opportunities are lost.

In this chapter, we aim to achieve a deeper integration of IE and inference with relational queries to improve both run-time efficiency and answer quality. We study two approaches.

We first consider deterministic Select-Project-Join (SPJ) queries over the ML results of Viterbi-based IE. For this case, we develop query optimizations that reduce work by carefully pushing relational operators into the Viterbi algorithm. Compared to previous approaches that treated the IE and query phases separately, these optimizations can significantly improve efficiency.

[1] In this Chapter, we use ML to stand for maximum-likelihood, rather than machine learning.

The key problem with querying the ML extractions stems from inaccuracies in IE models: even high-quality ML extractions contain errors, which can be exacerbated in SPJ queries over such extractions. We illustrate this point with an example scenario.

Example 10 *Kristjansson et al. [20] used CRF-based* IE *to extract* Contact *information from the signature blocks in the Enron email corpus [70]. One such block begins with the text:*

> *Michelle L. Simpkins*
> *Winstead Sechrest & Minick P.C.*
> *100 Congress Avenue, Suite 800...*

One of the fields we wish to extract from these email blocks is companyname, which in this case should be "Winstead Sechrest & Minick P.C.". Unfortunately, the ML extraction from the CRF model assigns the value NULL to companyname. This IE *error shows up more explicitly in the following relational queries:*

Query 1. Find contacts with companyname containing "Winstead"

```
SELECT *
FROM Contacts
WHERE companyname LIKE '%Winstead%'
```

Query 2. Find all contact pairs with the same companyname

```
SELECT *
FROM Contacts C1, Contacts C2
WHERE C1.companyname = C2.companyname
```

The first query provides an empty result, since "Winstead Sechrest & Minick P.C." is not identified as a companyname. The second query is perhaps more vexing and counterintuitive: the erroneous NULL parse for Michelle Simpkins' companyname means that she does not even match herself, hence, she is again absent from the output. In general, due to the inherent uncertainty in IE*, the answers to queries over ML extractions can contain false negatives and/or false positives.*

□

strID	apt. num^p	street num^p	street name^p	city^p	state^p	country^p
1	null	2181	Shattuck	North Berkeley	CA	USA
2	12B	331	Fillmore St.	Seattle	WA	USA
3	224B	null	Ford South St.	Louis	MO	USA

(a)

strID	company name^p	city^p	state^p
1	Google	Mountain View	CA
2	Yahoo!	Santa Clara	CA
3	Microsoft	Seattle	WA

(b)

Figure 6.1. Examples of the ML views of entity tables: (a) address strings (b) company location strings.

To address this answer quality problem, we develop an alternative to the ML-based approach using Probabilistic Database techniques, where *probabilistic* SPJ queries are computed over the set of possible worlds (PWs) induced from the CRF-based distribution.

We show that, by allowing us to look beyond ML extractions, this probabilistic database approach can significantly reduce false negatives (by as much as 80% in our experiments) with only marginal increase in false positives. As expected, the improvement in answer quality comes at a performance cost due to the consideration of the full CRF distribution instead of only the ML extraction. For probabilistic selection, we show how to reduce this overhead to a small fixed-cost per document over the optimized ML-based queries. For probabilistic join, however, the performance penalty can be much higher. Thus, the two approaches we develop, while both improvements over the state-of-the-art, provide a design space where performance or answer quality can be emphasized based on the requirements of the application.

In the following sections we first present novel optimizations for *SPJ queries over the maximum-likelihood world* of the CRF distribution that improve runtime performance through the effective integration of relational operators and Viterbi-style inference. Then, we propose new algorithms, that substantially extend Viterbi inference to enable effective *probabilistic SPJ queries over the full CRF distribution*, and thus provide improved answer quality by considering the full set of possible extraction "worlds". Finally, we evaluate our proposed approaches and algorithms, demonstrating the scalability of our solutions and exploring the trade-off between efficiency and answer quality using data collections from the Enron and DBLP datasets.

6.2 Two Families of SPJ Queries

This section describes the setup of the system. Unstructured text is represented by a set of documents or text-strings $\mathcal{D}$, and each document $d \in \mathcal{D}$ is represented by a set of token records in TOKENTBL. The CRF-based distribution over documents $\mathcal{D}$ is stored in MR table. The TOKENTBL and MR table are pre-materialized and has been described in Section 3.4. Based on TOKENTBL and MR, each document is parsed into one probabilistic record in an *entity table*. Two families of queries over the probabilistic entity tables are explored.

Entity Table: An entity table contains a set of probabilistic attributes, one for each label in Y. Each tuple in the entity table has an independent distribution, defined by the CRF model over the corresponding text-string d. Figure 6.1(a) shows the maximum-likelihood (ML) view of the entity table with three address strings.

The entity table is defined and generated by a *pivot operation* over the possible labelings in the TOKENTBL and their distribution. For each possible labeling of a text-string, the pivot operation generates one possible world of the corresponding record in the entity table. For example, the labeling corresponding to segmentation $\mathbf{y}_1$ in Figure 2.2(b) generates the first tuple in the entity table in Figure 6.1(a).

Two Families of SPJ Queries: There are two families of queries we consider computing on the extracted data in the entity tables. Given that the ML view of an entity table can be defined as:

```
CREATE VIEW entityTbl1-ML as
  SELECT *, rank() OVER (ORDER BY prob(*) DESC) r
  FROM   entityTbl1
  WHERE  r = 1;
```

one family is the set of deterministic SPJ queries over the ML views of the entity tables. The other family computes the top-k results of probabilistic SPJ queries over entity tables using the set of possible worlds (i.e., segmentations) induced by the CRF distribution. Optionally, a prescribed threshold can be applied to the result probabilities. The queries in this family have the following general form, where `SQLQuery` is a view using a standard SPJ query:

```
SELECT *, rank() OVER (ORDER BY prob(*) DESC) r
FROM   SQLQuery
WHERE  r <= k [ AND prob(*) > threshold ]
```

6.3 Querying the ML World

In this section, we focus on deterministic SPJ queries over the ML views of the entity tables. The naive way to compute this type of query is to first compute the ML extraction for each document, then execute the SPJ queries over the result. We show that for selection and join, conditions can be pushed into the Viterbi inference process to achieve significant speed-up. This performance improvement demonstrates the benefit of a deep integration of IE and relational operators.

6.3.1 Optimized Selection over ML World

An example of a selection query over the ML view of an entity table is to find all the *Address* tuples whose *streetname* contains 'Sacramento' in the ML extraction:

```
SELECT *
FROM Address-ML
WHERE streetname like '%Sacramento%'
```

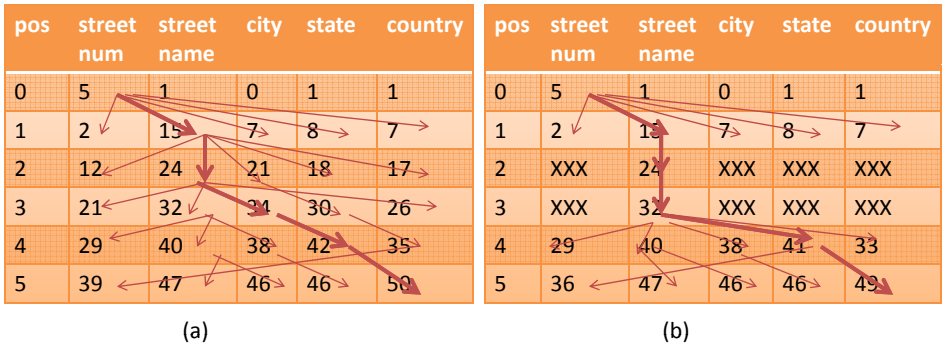


Figure 6.2. Illustration of the computation of V matrix in the following algorithms: (a) Viterbi; (b) ConstrainedViterbi.

The selection condition over the ML view of the *Address* table is rewritten into two selection conditions over the underlying TOKENTBL and MR: (1) test if the text-string *d* contains the token sequence in the selection condition x_{cond} (e.g., *'Sacramento'*); (2) test if the position(s) where x_{cond} appears in *d* are assigned the label in the selection condition y_{cond} (e.g., *streetname*). The first condition can be simply pushed down over the TOKENTBL, only picking the documents that contain the token sequence x_{cond}. The second condition can be pushed into the Viterbi inference computation of a particular text-string *d* with condition $\{i, len, y'\}$ ensuring that the label at positions from i to $(i + len - 1)$ is y'. In general, condition (2) may span multiple tokens. Next, we describe SELVITERBI, a novel variant of the Viterbi algorithm that pushes the condition $\{i, len, y'\}$ into the Viterbi computation.

Recall the Viterbi algorithm described in Section 5.3.1, which computes a V matrix as in Figure 6.2(a): The condition $\{i, len, y'\}$ is satisfied if and only if the ML segmentation path backtracks to all the entries in $V(i, y')$ .. $V((i + len - 1), y')$. The intuition behind the SELVITERBI algorithm is as follows. Given condition $\{i, 1, y'\}$ and a text-string *d*, if none of the top-1 (partial) segmentations ending at position j where $T \geqslant j > i$ backtrack to cell $V(i, y')$, then we can stop the inference and conclude that this text-string does not satisfy the condition. We call such a position j a *pruning position*, and we would like to detect a pruning position as early as possible to stop the inference computation. Similar intuition holds for the general case where $len > 1$. Our SELVITERBI algorithm follows the basic Viterbi dynamic program algorithm, which calls a User Defined Function (UDF), at each recursive step i, to compute the top-1 (partial) segmentations in $V(i, y_i)$ using Equation (2.2), with additional logic to check if i is the smallest pruning position. It stops the recurrence and returns *null* if a pruning position is found. The pseudo-code of the SELVITERBI algorithm is shown in Figure 6.3 and 6.4.

Example 11 *Using the example in Figure 6.2(a), suppose the condition* $\{1, 1,$*streetnumber*$\}$ *is generated from a selection condition for a specific text-string d: return text-string d with streetnumber as the label of token* $\#2$ *(counting from* 0*) in the ML segmentation. In the second recursive step, only the partial segmentation in* $V(1,$*streetnumber*$)$ *satisfies the condition. In the third recur-*

```
1 CREATE FUNCTION selViterbi (id, i, len, y') RETURN VOID AS
2 $$
3 -- compute the V matrix from tokenTbl and factorTbl
4 INSERT INTO V
5 WITH RECURSIVE Vtop1(pos, entries, filter) AS (
6      SELECT st.pos,
7         mr.score[1:|Y|]||array-fill(-1,ARRAY[|Y|]) entries,
8         array-fill(0,ARRAY[|Y|]) filter
9      FROM tokenTbl st, factorTbl mr
10     WHERE st.strID=id AND st.pos=0 AND mr.segID=st.segID
11     UNION ALL
12     SELECT st.pos,
13        top1-array(v.score, mr.score) entries,
14        update-filter(i,len,y',v.pos,v.filter,v.score) filter
15     FROM tokenTbl st, Vtop1 v, factorTbl mr
16     WHERE st.strID=id AND st.pos = v.pos+1 AND
17        mr.segID=st.segID AND (pos<=i or nonzero(v.filter))
18 ) SELECT pos, entries FROM Vtop1,
19 -- backtrack the ML segmentation from V
20 INSERT INTO Ptop1
21 WITH RECURSIVE P(pos,entry) AS (
22      SELECT pos, get-max(entries) entry
23      FROM Vtop1 WHERE pos=T-1
24      UNION ALL
25      SELECT V.pos, V.entries[P.prevLabel]
26      FROM Vtop1 V, P WHERE V.pos = P.pos-1
27 ) SELECT id as strID, pos, entry.label FROM Ptop1
28 $$
29 LANGUAGE SQL;
```

Figure 6.3. SELVITERBI(): optimized algorithm for select-over-ML queries, which stores the ML segmentation in Ptop1.

UPDATE-FILTER $(i, len, y', j, filter, V(j, y))$

1 **for** *each* $l \in Y$ **do**

2 **if** $j \geqslant (i + len) \&\& filter[V(j, l).prev] = 1$ **then**

3 $newFilter[l] = 1;$

4 **else if** $j > i \&\& l = y' \&\& filter[V(i-1, l).prev] = 1$ **then**

5 $newFilter[l] = 1;$

6 **else if** $j = i \&\& l = y'$ **then**

7 $newFilter[l] = 1;$

8 **else** $newFilter[l] = 0;$

9 **endif endfor**

10 **return** $newFilter$

Figure 6.4. UPDATE-FILTER(): auxiliary function for checking pruning position.

sive step, because no partial segmentations in $V(2, y), y \in Y$ *come from cell* $V(1,$*streetnumber*$)$ *as shown in Figure 6.2(a),* $j = 2$ *is the smallest pruning position. Thus, we stop the dynamic programming algorithm and conclude that d does not satisfy the condition* $\{1, 1,$*streetnumber*$\}$. □

Variants of SELVITERBI can be employed for increased efficiency based on criteria such as the position and selectivity of a selection condition (which can be incorporated into a cost-based optimizer).

Generalized Selection-Aware Viterbi: The SELVITERBI algorithm can be used with any $\{i, len, y'\}$ condition. However, for conditions with i being the last few positions, a variant of the Viterbi algorithm, which computes the top-k segmentations from the end to the start of a text-string, should be used for efficiency. Furthermore, the Viterbi algorithm can be modified to start the dynamic programming from any position i, by expanding the intermediate result of the partial segmentations in two directions, rather than just one (left to right) in Viterbi. As future work, a cost-based optimizer can be developed to decide where/how to execute SELVITERBI, based on the position and selectivity of a selection condition.

6.3.2 Optimized Join over ML World

An example of a join query over the ML views of two entity tables is to find all (*Address*, *Company*) entity pairs (as in Figure 6.1) with the same *city* value.

```
SELECT *   FROM Address-ML, Company-ML
WHERE Address-ML.city = Company-ML.city
```

The naive join algorithm over the ML world is: first, compute the Viterbi inference on the two input document sets independently; second, perform the pivot operations to compute the ML views of the *Address* and *Company* entity tables – *Address-ML* and *Company-ML*; finally, compute the deterministic join over the *city* attribute.

The intuition behind the optimization is that we do not need to compute the Viterbi inference over a text-string in the outer document set if it contains none of the join-key values computed from the inference over the inner document set. The optimized algorithm for join-over-ML follows these steps: (1) compute the Viterbi inference over the smaller of the two input document sets, we call this the inner set and the other the outer set; (2) build a hash-table of the join-attribute values computed from the ML extraction of the inner set; (3) perform Viterbi inference only on the documents in the outer set that contain at least one of the hashed values; and (4) perform the pivot operation to compute the ML views of the entity tables and compute the deterministic join over them. This optimization reduces the number of documents on which the Viterbi inference is performed, which improves the performance of join-over-ML queries (by more than a factor of 5 in our experiments).

6.4 Querying the Full Distribution

In this section, we discuss novel probabilistic query processing strategies to compute the top-k results of SPJ queries over the possible worlds of CRF-based entity tables. In contrast to the techniques in Section 6.3, the algorithms here do not focus on the single ML segmentation, but rather compute results from the CRF-induced distribution of possible segmentation worlds. As a first step, we present an *incremental* Viterbi algorithm that supports efficient sorted access to the possible segmentations of a document by descending probabilities. This ordering is critical for efficiently computing the top-k results of SPJ queries, especially probabilistic joins.

6.4.1 Incremental Viterbi Inference

The conventional Viterbi top-k inference algorithm is a straightforward adaptation of the dynamic programming recurrence in Equation (2.2). The idea is to maintain, in each $V(i, y)$ cell, a list of the top-k partial segmentations ending at position i with label y. This, of course, assumes that the "depth" parameter k is known a priori.

We present a variant of the Viterbi algorithm that can *incrementally* (through a GET-NEXT-SEG() call) compute the next highest-probability segmentation of a document. Our algorithm is similar in spirit to the theoretical framework of Huang et al. [71] for finding k-best trees in the context of hypergraph traversals.

Recall the V matrix computed by the Viterbi algorithm, where each cell $V(i, y)$ contains a list of top-k entries $e = \{score, prev(label, idx)\}$ ordered by $score$. The key idea in our incremental algorithm is to enumerate this list lazily: only on the path of the last extracted segmentation.

The key idea in our incremental algorithm is to backtrack the last extracted segmentation and use the basic Viterbi formula (Equation (2.2)) to compute the next highest-probability segmentation. Since all the $V(T)$ (T is the length of the document) entries in already-extracted (i.e., "consumed") segmentation paths cannot be used to compute the next highest-probability segmentation, they recursively trigger (through a GET-NEXT() call) the computation for the next "unconsumed" entry using Equation (2.2) from their predecessor V cells. The key observation here is that these recursive GET-NEXT() calls can only occur *along the path of the last extracted segmentation*. More specifically, assume y^* denotes the ML segmentation discovered through Viterbi. To build the next-best segmentation, we start from entry $V(T, y^*[T])$. Since that entry is already "consumed" (in the ML segmentation), we trigger a GET-NEXT() call to compute the next-best entry using the entries in row $V(T-1)$; this, in turn, can trigger a GET-NEXT() call at $V(T-1, y^*[T-1])$, and so on. Thus, GET-NEXT() calls are recursively invoked on the path y^* from T down to 1 (or, until we find a $V(i, y^*[i])$ cell with an already-computed "unconsumed" entry).

The incremental Viterbi algorithm in Figure 6.5 takes the document doc, the current V matrix, the last top segmentation y^*, and the number of top segmentations generated so far L, to compute the next highest-probability segmentation y^*_{new}. If $L = 0$, then GET-NEXT-SEG() calls VITERBI() to compute the ML segmentation. If $L > 0$, then recursive GET-NEXT() calls are triggered for V cells on the y^* path. The chain of GET-NEXT() calls stops either at row 1 or when $y^*[i]$ is not the last entry in cell $V(i, y^*[i])$. Equation (2.2) is used to compute the next probable (partial) segmentation in $V(i, y^*[i])$ after computing the new entry for $V(i-1, y^*[i-1])$. Finally, GET-NEXT-SEG() backtracks the next highest-probability segmentation y^*_{new} from the $(L+1)$th maximal entry in row $V(T)$.

Complexity: When $L = 0$, GET-NEXT-SEG() calls VITERBI(), thus the complexity is $O(T|Y|^2)$. When $L > 0$, the complexity of the incremental Viterbi algorithm GET-NEXT-SEG() is $O(T(|Y| + L)log(|Y| + L))$, because GET-NEXT() is called maximum T times, where T is the length of the document. Each GET-NEXT() call needs to sort at most $(|Y| + L)$ entries, because each cell in $V(i, y_i)$ contains 1 entry after the VITERBI() ML inference, and at most 1 entry is added to 1 cell in $V(i, y_i)$ for each GET-NEXT-SEG() call.

Example 12 *The Viterbi V matrix in Figure 6.2(a) shows the ML segmentation in bold. In order to compute the second highest-probability segmentation,* GET-NEXT*() is triggered recursively from* GET-NEXT-SEG*() to compute the next probable (partial) segmentations entry for cells V(5,*country*), V(4,*state*), V(3,*city*), V(2,*streetname*) and V(1,*streetname*) on the ML path. The second highest-probability segmentation is then computed by backtracking the entry in $V(T, y_T)$ with the second highest score.* □

GET-NEXT-SEG (doc, V, y^*, L)

1 **if** $L = 0$ **then**

2 $y^*_{new} \leftarrow$ VITERBI(doc);

3 **else**

4 $T \leftarrow doc.length$;

5 *// compute the next best entry for cell* $V(T, y^*[T])$

6 GET-NEXT$(V, T, y^*[T])$;

7 $Tmp \leftarrow \cup_{y_T \in Y} V(T, y_T)$;

8 *sort* Tmp *by score desc*;

9 $y^*_{new}[n] \leftarrow Tmp[L + 1]$;

10 **for** $i = n - 1; i > 0; i - -$ **do**

11 $y^*_{new}[i] \leftarrow y^*_{new}[i + 1].prev$;

12 **endfor endif**

13 **return** y^*_{new}

GET-NEXT (V, i, e)

1 $maxidx \leftarrow V(i, e.label).length; l \leftarrow e.label$;

2 **if** $e = V(i, l)[maxidx]$ **then**

3 GET-NEXT$(V, i - 1, y^*[i - 1])$;

4 $Tmp \leftarrow \cup_{y_{i-1} \in Y} V(i - 1, y_{i-1})$;

5 **for** *each* $te = \{score, label, prev\} \in Tmp$ **do**

6 $te.score \leftarrow te.score +$ MR$(l, te.label, x_i)$;

7 **endfor**

8 *sort* Tmp *by score desc*;

9 $V(i, l) \leftarrow V(i, l) \cup Tmp[maxidx + 1]$;

10 **endif**

Figure 6.5. The incremental Viterbi algorithm for getting the next highest-probability segmentation.

6.4.2 Probabilistic Selection

The following query computes the top-1 result of probabilistic selection with condition *streetname contains 'Sacramento'* over the probabilistic *Address* entity table:

```
SELECT *, rank() OVER (ORDER BY prob(*) DESC) r
FROM (  SELECT *
        FROM Address
        WHERE streetname like '%Sacramento%'
) as Address
WHERE r = 1 AND prob(*) > threshold
```

Setting a `threshold` on probability can effectively filter out false positive results with very low probabilities that could be introduced by considering additional non-ML worlds. For example, for string "123 Fillmore Street Sacramento CA", the above query should not return the erroneous extraction where "Sacramento" is labeled *streetname*. Fortunately, the CRF model exploits the "context" (e.g., position, previous label) of a token, so in the "context" of string "123 Fillmore Street Sacramento CA", "Sacramento" has a very low probability of being *streetname*, which can be filtered out in the selection query with proper setting of the `threshold`. Please refer to Section 6.5.1 for a discussion on `threshold` setting.

The probabilistic selection condition is translated into domain constraints that the labels at the positions where x_{cond} (e.g., "Sacramento") appear in the text-string *d* can only be y_{cond} (e.g., *streetname*). For example, with string "123 Sacramento Street San Francisco CA" and condition *streetname contains 'Sacramento'*, the domain constrains are $D(y_2) = \{\textit{streetname}\}$ and $D(y_i) = Y$ for $y_1, y_3, .., y_6$. The domain constraints $D(y_1), ..., D(y_n)$ generated from the probabilistic selection condition are used as input to the *constrained Viterbi* algorithm, described in Section 2.2.1, to compute the top-1 result of probabilistic selection. We illustrate the algorithm using the following example.

Example 13 *Consider the V matrix in Figure 6.2(b) with domain constraints such that y_2 and y_3 can only be streetname: $D(y_2) = D(y_3) = \{\textit{streetname}\}$. Thus, we can cross out all the cells in $V(2)$ and $V(3)$ where $y \neq$ streetname as shown in Figure 6.2(b). These domain constraints result in a top-1 constrained path that is different from the result of the basic Viterbi algorithm. The top-1 constrained path satisfies the selection condition, and will be returned as a result if its probability is higher than a certain threshold. This constrained top-1 path is the non-ML extraction computed for the probabilistic selection queries.* □

6.4.3 Probabilistic Join

The following query computes the top-1 (ML) result of a probabilistic join operation over the probabilistic entity tables *Address* and *Company*, with an equality join condition on *city*.

```
SELECT *, rank() OVER (ORDER BY prob(*) DESC) r
FROM (  SELECT *
        FROM Address A, Company C
        WHERE A.city = C.city
) as AddrComp   WHERE r = 1
```

A naive probabilistic join algorithm is: first compute the top-k extractions (for a sufficiently large k) from each input document then deterministically join over the top-k views of the entity tables. The complexity of this algorithm is dominated by the expensive computation of the top-k extractions. Computing a fixed number of top-k extractions for every input document is wasteful, because most top-1 probabilistic join results can be computed from the first few highest-probability extractions. Thus, a more efficient way is to compute the top-1 join results incrementally.

An incremental join algorithm is the rank-join algorithm described in [72]. It takes two ranked inputs and computes the top-k join results incrementally without consuming all inputs. The algorithm maintains a priority queue for buffering all join results that cannot yet be produced, and an upper-bound of the score of all "unseen" join results. Using the *buffer* and the *upper-bound*, the rank-join algorithm can decide if one of the produced join results is guaranteed to be among the top-k answers.

The key idea behind our probabilistic join algorithm is that, given the incremental Viterbi algorithm, which gives us ordered access to the possible extractions of a string *d*—a ranked list by probability—we can compute the top-1 join result for a pair of strings using the rank-join algorithm. Thus, our probabilistic join is a *set of rank-join computations*—one for each string pair that is "joinable" (i.e., can potentially lead to join results). If most "joinable" string pairs compute the top-1 most probable join result from the top-1 extractions, then GET-NEXT-SEG() is called far fewer than k times per document. For the remaining "joinable" string pairs, more extractions are computed incrementally to look for join results in non-ML worlds. In order to bound the search, we set k to be the maximum number of extractions from a document.

If a string pair does not join within the top-k extractions, then rank-join incurs extra overhead. Thus, an effective filter for non-"joinable" string pairs is crucial to the efficiency of the probabilistic join algorithm. We find that an effective such filter is to first compute the top-k extractions for inner (the smaller) entity table, build a hash-table for all the join-key values, and for each string in outer, probe the join-key hash-table for the "joinable" inner strings.

The probabilistic equijoin algorithm in Figure 6.6 takes two joining relations: $inner$, $outer$, two joining columns: $incol$ and $outcol$, and a parameter k indicating an upper bound on the number of extractions allowed per document. The algorithm first computes the top-k segmentations for each $inner$ document, which overlaps with at least one document in $outer$, using the incremental Viterbi algorithm.

For each string $odoc \in outer$, which contains at least one join-key from an $inner$ extraction, we compute the corresponding $joinable$ set—a set of inner documents that have a join-key contained in $odoc$. Then, the rank-join loop starts simultaneously for all $idoc \in joinable$ and $odoc$ pairs. The next highest-probability segmentation $yout^*$ for $odoc$ is computed. All $inner$ segmentations that have join-key value $yout^*.outcol$ are retrieved, and the joined results are inserted into $matched$—a hash-table containing all the matching $\{idoc, odoc\}$ pairs and their corresponding rank-join buffer containing matching segmentations $\{yin^*, yout^*\}$. Next, the upper-bound $upper$ for each $\{idoc, odoc\}$ pair in $matched$ is computed, and the segmentation pairs $\{yin^*, yout^*\}$ that have probability no less than $upper$ are returned. Finally, $idoc$ and $\{idoc, odoc\}$ are deleted from $joinable$ and $matched$, respectively. The rank-join loop stops when either the $joinable$ set becomes empty or the extraction depth exceeds k.

6.4.4 Probabilistic Projection

The following projection query computes the ML *city* value for each record in the entity table *Address*.

```
SELECT *, rank() OVER (ORDER BY prob(*) DESC) r
FROM (  SELECT city
        FROM Address  ) as Address
WHERE r = 1
```

A naive strategy would be to first compute the full result distribution of a *partial marginalization* over a sub-domain $\chi \subset Y$ for each label y_i in the CRF model (using the distribution of all possible segmentations) [2]; then, the ML world from the distribution can be returned. In this manner, however, significant effort is wasted in calculating the non-ML worlds of the marginal distribution.

Next, we describe a novel probabilistic projection algorithm that integrates the top-1 Viterbi inference and the marginal inference. The key idea underlying our probabilistic projection technique is to extend the Viterbi dynamic program to compute top-1 segmentation paths with labels that are either projected on (i.e., in $Y \backslash \chi$) or *'don't care'*s marginalizing over all the projected-out labels in chi.

Similar to basic Viterbi, our probabilistic projection algorithm (for projecting out a sub-domain $\chi \subset Y$) also follows a dynamic programming paradigm, computing ML (partial) segmentations ending at position i from those ending at position $i-1$. There are, however, two crucial differences. First, the (partial) segmentations $\{y_1, ..., y_i\}, i \leqslant T$ computed for probabilistic projection only have labels $y_j, 1 \leqslant j \leqslant i$ in $Y \backslash \chi$ or *'don't care'*. $y_j =$*'don't care'* means that the segmentation path is marginalized over χ at position j (i.e., y_j could be *any* label in χ). A dynamic programming matrix $V(i, y)$ stores such top-k (partial) segmentations that end in label $y \in Y$ at position i. Second, in order to compute the top-k (partial) segmentations in $V(i+1, y_{i+1})$, in addition to the $V(i, y_i)$ entries, we also need to account for the possibility of a *'don't care'* in position i. This is

[2] Note that, in contrast to conventional marginal inference (Section 2.2.1), which marginalizes to eliminate RVs in $\mathbf{y}$, partial marginalization here marginalizes to *restrict the domains* of RVs in $\mathbf{y}$.

TOP1PROBJOIN $(inner, outer, k, incol, outcol)$

1 **for** *each* $idoc \in inner$ *that overlap with at least one* $odoc \in outer$ **do**

2 $V \leftarrow null; yin^* \leftarrow null;$

3 **for** $i = 0; i < k; i$**++ do**

4 $yin^* \leftarrow$ GET-NEXT-SEG$(idoc, V, yin^*, i);$

5 $inTopk$.PUT$(\{idoc, i\}, yin^*); inKeys$.PUT$(yin^*.incol, \{idoc, i\});$

6 **endfor endfor**

7 **for** *each* $odoc \in outer$ *that contains at least one* $inKeys.keys$ **do**

8 $V \leftarrow null; yout^* \leftarrow null;$

9 $joinable \leftarrow$ *all* $idoc$ *that has* $inKeys.keys$ *contained in* $odoc$

10 **for** $i = 0; i < k; i++$ **do**

11 $yout^* \leftarrow$ GET-NEXT-SEG$(odoc, V, yout^*, i); outTopk$.PUT$(\{odoc, i\}, yout^*);$

12 **for** *each* $\{idoc, idx\} \in inKeys$.GET$(yout^*.outcol)$ **do**

13 **if** $idoc \in joinable$ **then**

14 $yin^* \leftarrow inTopk$.LOOKUP$(\{idoc, idx\}); matched$.PUT$(\{idoc, odoc\}, \{yin^*, yout^*\})$

15 **endif endfor**

16 **for** *each doc pair* $\{idoc, odoc\} \in matched$ **do**

17 $upper \leftarrow outTopk(odoc).maxprob \times inTopk(idoc).minprob;$

18 $upper \leftarrow \max(yout^*.prob \times inTopk(idoc).maxprob, upper);$

19 $prob \leftarrow yin^*.prob \times yout^*.prob$

20 **if** $prob \geqslant upper$ **then**

21 **return next** $\{idoc, odoc, yin^*, yout^*\}$

22 $matched$.DELETE$(\{idoc, odoc\}); joinable$.DELETE$(idoc);$

23 **endif endfor**

24 **if** $joinable = \emptyset$ **then** $break;$

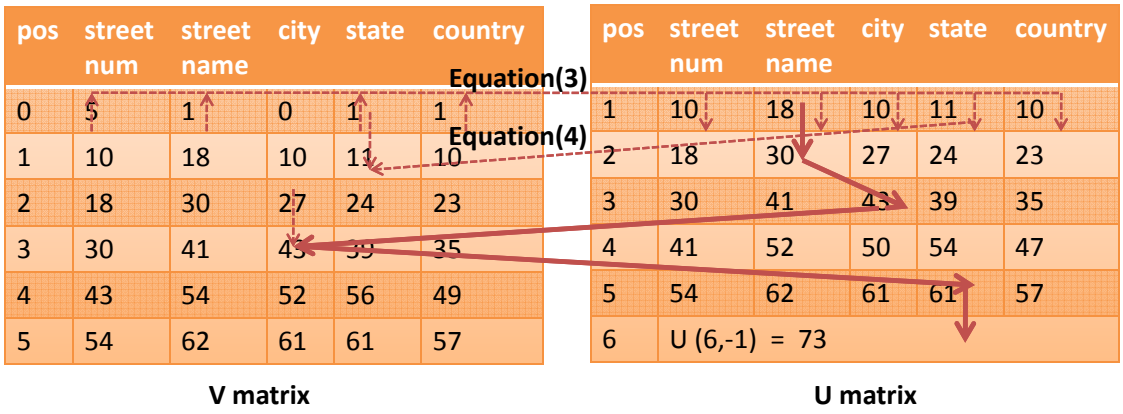
25 **endif endfor endfor**

Figure 6.6. Algorithm for computing the top-1 result for probabilistic join queries.

$$U(i+1, y_{i+1}).score = \log(\max_{prev}\{\Sigma_{y_i \in \chi}(\bowtie_{prev} (exp(V(i, y_i).score + FactorTbl(x_{i+1}, y_i, y_{i+1}))))\}) \quad (6.1)$$

$$V(i+1, y_{i+1}).score = \max\{\max_{y_i \in Y \setminus \chi}\{(V(i, y_i).score + FactorTbl(x_{i+1}, y_i, y_{i+1}))\}, U(i+1, y_{i+1}).score\}$$

Figure 6.7. Equations for computing U and V matrix for probabilistic projection algorithm.



pos	street num	street name	city	state	country
0	5	1	0	1	1
1	10	18	10	11	10
2	18	30	27	24	23
3	30	41	43	39	35
4	43	54	52	56	49
5	54	62	61	61	57

pos	street num	street name	city	state	country
1	10	18	10	11	10
2	18	30	27	24	23
3	30	41	43	39	35
4	41	52	50	54	47
5	54	62	61	61	57
6	U (6,-1) = 73				

Figure 6.8. Illustration of the data structures in probabilistic projection.

accomplished by explicitly maintaining an auxiliary matrix $U(i+1, y_{i+1})$ that stores the (partial) top-k segmentations ending at position $i+1$ with label $y_{i+1} \in Y$ and $y_i =$ *'don't care'*. Example V and U matrices for the above example probabilistic projection query are shown in Figure 6.8.

Next, we describe how the entries in the V and U matrices are computed. Our probabilistic projection algorithm is a variation of the Viterbi technique, where a dynamic programming recurrence computes both V and U simultaneously. Equation (4) in Figure 6.7 computes the top-1 partial segmentation in cell $V(i+1, y_{i+1})$ by selecting either the best extension of a partial segmentation in $V(i, y_i), y_i \in Y \setminus \chi$ or $U(i+1, y_{i+1})$, depending which choice gives the maximum score. Equation (3) in Figure 6.7 computes the top-1 partial segmentation for cell $U(i+1, y_{i+1})$, marginalizing over χ at position i, given label $y_{i+1} \in Y$. If we think of each $V(i, y), y \in Y$ as a ranked list of tuples $\{score, label, prev\}$ ordered by descending $score$, then computing the top-ranked $U(i+1, y_{i+1})$ entry is a *multi-way rank-join* with an equality join condition on $prev$ over ranked lists $V(i, y), y \in \chi$. The multi-way rank-join aggregates the entries (i.e., partial segmentations) in multiple lists $V(i, y), y \in \chi$, which agree on $prev$. The join condition (equality on $prev$) ensures that the joined (partial) segmentations share the same path up to position $i-1$. The $score$ of the resulting partial segmentation is the summation over the $score$'s of the joined partial segmentations in $V(i, y), y \in \chi$.

The top-1 result for probabilistic projection can be computed by backtracking the $prev$ pointers from either an entry in $V(T, y_T), y_T \in Y \setminus \chi$ or an aggregated entry $U(T+1, -1)$ computed by aggregating $V(T, y_T), y_T \in \chi$ (to account for a final label of *'don't care'*).

The detailed pseudo-code descriptions of probabilistic projection algorithm can be found in Appendix A.2.

Complexity: The complexity of the probabilistic projection algorithm is $O(T^3|Y|^3 l)$, where l is the average depth of the rank-join algorithm in GET-NEXT-U(). The additional complexity comes from computing the U matrix. The computation of each entry in the $U(i, y_{i+1})$ matrix may recursively triggers up to T number of GET-NEXT-U() calls. And each GET-NEXT-U() is a rank-join algorithm, which calls MARGINALIZATION() with complexity $O(T|Y|^2)$ in each iteration. Thus, we have the complexity being $O(T^3|Y|^3 l)$.

Example 14 *Suppose the query is probabilistic projection over city on Address. Figure 6.8 illustrates the computation of the entries in the V and U matrices. Equation (3) is used to compute $U(i+1, y'), y' \in Y$ matrix entries from a rank-join over all the $V(i, y)$ lists except when $y =$city. Equation (4) is used to compute $V(i+1, y)$ matrix entries from the maximum of $V(i, city)$ and $U(i+1, y)$. The top-1 result of probabilistic projection is backtracked from the maximum of $V(5, city)$ and $U(6, -1)$ ($T = 5$) through $prev$ pointers. In this example, the most probable segmentation after probabilistic projection (i.e., partial marginalization) is {'don't care','don't care','don't care',city,'don't care','don't care'}.*

6.5 Experimental Results

Having described two families of approaches for integrating query processing and IE, we now present the results of a set of experiments aimed to evaluate them in terms of efficiency and answer quality.

Setup and Dataset: We implemented the optimizations for select-over-ML (sel-ML) and join-over-ML (join-ML) queries as described in Section 6.3, and the algorithms for computing the top-1 result for probabilistic selection (prob-sel) and probabilistic join (prob-join) as described in Section 6.4. These implementations were done on PostgreSQL 8.4.1. We conducted the experiments reported here on a 2.4 GHz Intel Pentium 4 Linux system with 1GB RAM.

For the accuracy experiments, we use the Contact dataset [73] and the CRF model developed at the University of Massachusetts [20]. The Contact dataset contains 182 signature blocks from the Enron email dataset annotated with contact record tags (e.g., *phonenumber*, *city*, *firstname*, etc.). For ground truth, we rely upon manual tagging performed in prior work [20]. We use false positives (i.e., the number of computed results not in the ground truth) and false negatives (i.e., the number of results in ground truth not computed) as the two measures of accuracy.

For the efficiency and scalability experiments we use the DBLP dataset [69] and a CRF model with similar features to those of [20]. The DBLP database contains more than $700k$ papers with

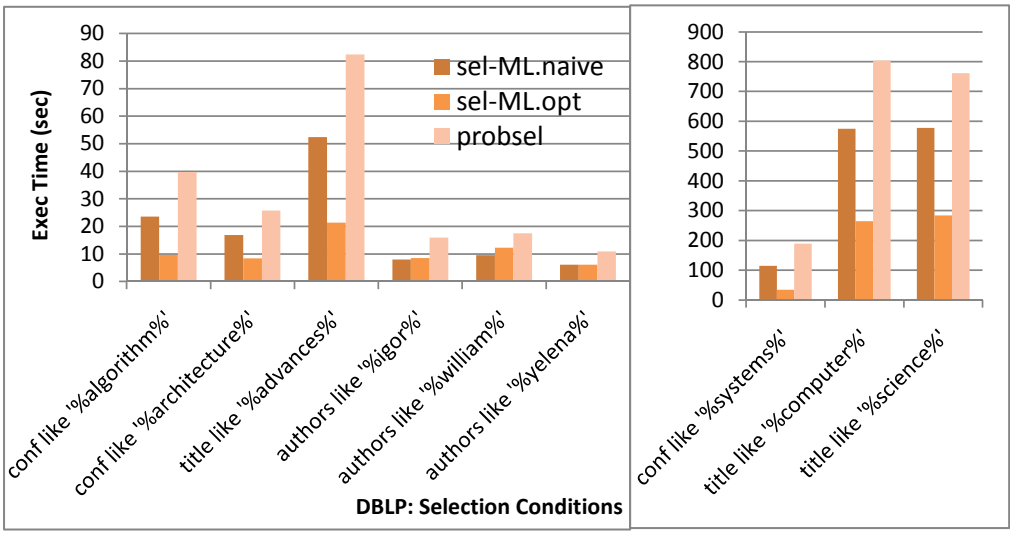


Figure 6.9. Performance comparison (DBLP) between sel-ML.naive, sel-ML.opt and prob-sel with difference selection conditions.

attributes, such as *conference*, *year*, etc. We generate bibliography strings from DBLP by concatenating all the attribute values of each paper record. Perhaps because it was generated from structured data, the ML extraction accuracy on the DBLP dataset is as high as 97%, making it not useful for the accuracy experiments. But it serves as an excellent stress test for the scalability of the algorithms.

6.5.1 Selection over ML world (sel-ML): opt vs. naive

In the first experiment, we use the DBLP dataset to compare the efficiency of optimized and naive algorithms for select-over-ML over $100k$ randomly picked documents (there is no answer quality difference between them — they provide exactly the same answers). We expect the optimized algorithm sel-ML.opt (using SELVITERBI()) to outperform the naive algorithm sel-ML.naive for selection conditions on token sequences that can have different labels in different documents (e.g., 'algorithm' can be in a paper *title* or a *conference* name). Figure 6.9 shows performance results for sel-ML.naive and sel-ML.opt as the left two bars of each selection condition. It can be seen that the optimized approach can achieve a speedup of 2 to 3 times for 6 selection conditions in Figure 6.9 [3].

We also ran experiments comparing the two approaches for cases where there is no expected benefit to the sel-ML.opt algorithm (i.e., where there is no pruning position for sel-ML.opt to exploit). As we can see in the three middle selection conditions in Figure 6.9, the performance of the two approaches was nearly identical, demonstrating that the optimized approach imposes negligible overhead (10% in the worst case) on the selection algorithm.

[3] It is 10-fold more expensive to evaluate the three right selection conditions, because they appear much more frequently in our dataset.

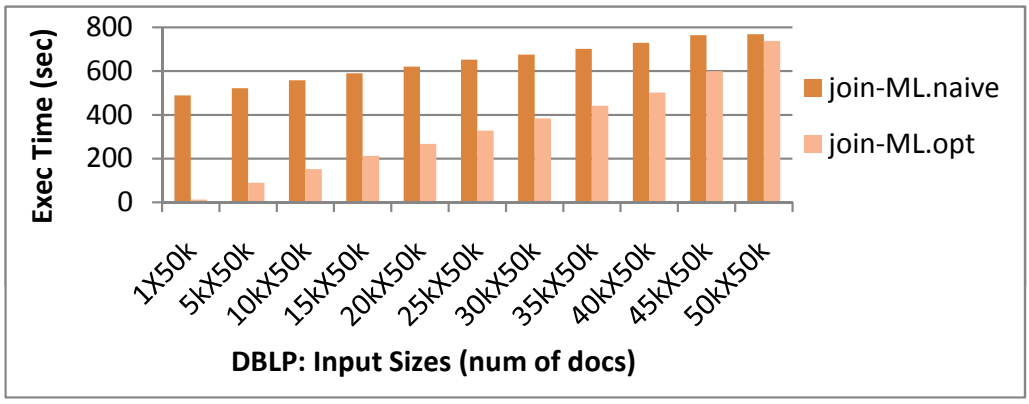


Figure 6.10. Performance comparison (DBLP) between join-ML.naive and join-ML.opt with different input sizes.

6.5.2 Join over ML World (join-ML): opt vs. naive

In the second experiment, we compare the efficiency of the optimized and naive algorithms for join-over-ML. We ran a query over two sets of documents from DBLP to find all pairs in the same *proceeding*. Figure 6.10 shows the results for different input sizes ($x \times y$ denotes a join of a set of x documents as the inner with a set of y documents as the outer). As can be seen in the figure, join-ML.opt is more than 5 times faster than join-ML.naive when the input sizes are $5k \times 50k$. The speedup of join-ML.opt decreases as the inner size approach the outer size.

Two statistics determine the relative performance of join-ML.opt compared to join-ML.naive: the number of join-keys computed from the inner, and the selectivity of those join-keys over the outer. The query joining on "*proceeding*" (shown in Figure 6.10) is a good case, where it generates a small set of join-keys, which in turn have low selectivity. As the inner size increases, both the number of join-keys and the selectivity increases, hence the speedup decreases.

Next, we ran a query over two sets of documents from the DBLP to find all pairs with the same *authors*. Figure 6.11(a) shows the results for different input sizes. As can be seen in the figure, join-ML.opt is more efficient than join-ML.naive when the inner size is much lower than outer size, and join-ML.opt grows to be more expensive as the inner size increases. For this join query, the size of inner join-keys is large and the selectivity of those join-keys is high.

Similarly, we ran the query joining on *publisher*. Since there are not many publishers, the size of the inner join-keys remain small as the size of the inner increases, while the selectivity of those join-keys is high. Thus, as can be seen in Figure 6.11(b), join-ML.opt have similar or slightly better performance compare to join-ML.naive for all inner sizes.

There are cases, when the cost of filtering the outer documents using the inner join-keys outweigh the cost of inference over the filtered documents, the join-ML.opt becomes more expensive than join-ML.naive. A query optimizer can estimate the statistics on the size and selectivity of the inner join-keys and determine when it is beneficial to apply join-ML.opt.

In the previous two sections, we showed that the optimizations for SPJ-over-ML queries can

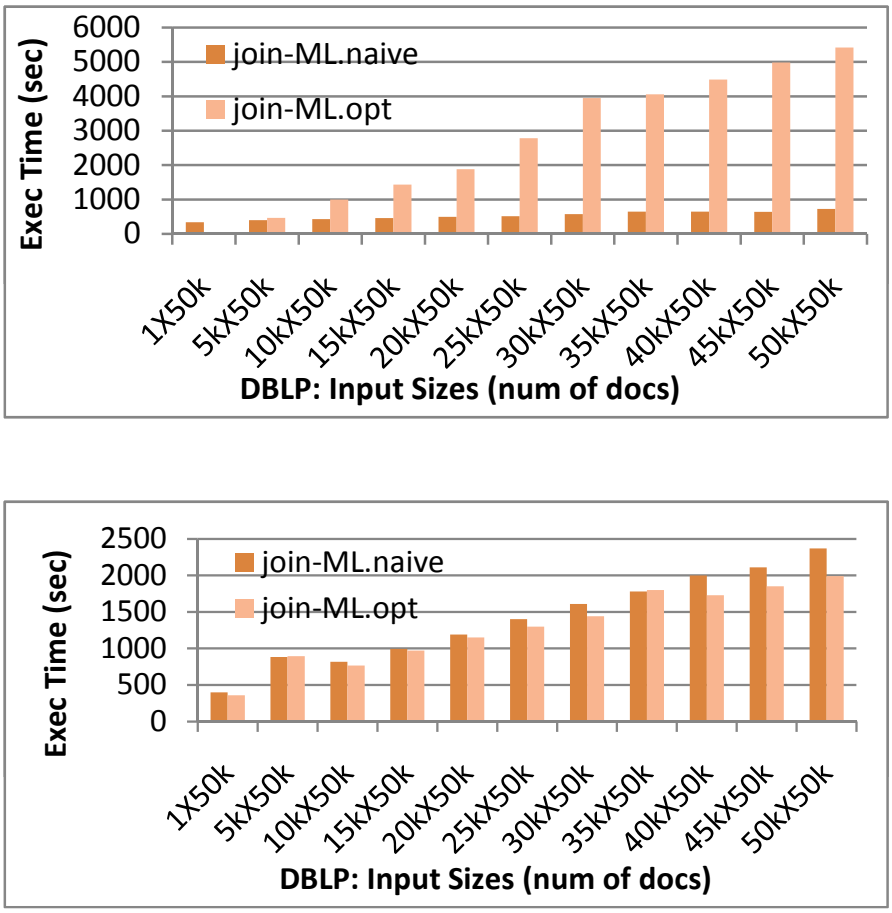


Figure 6.11. Performance comparison (DBLP) between join-ML.naive and join-ML.opt with different input sizes.

(False -)	company	firstname	lastname	jobtitle	department
sel-ML	0.074	0.012	0.036	0.102	0.286
prob-sel	0.014	0	0.006	0.037	0.079
(False +)	company	firstname	lastname	jobtitle	department
sel-ML	0.010	0	0	0.010	0
prob-sel	0.009	0.006	0.006	0.010	0

Figure 6.12. False negative, false positive rates (Contact) between sel-ML.opt and prob-sel queries.

achieve significant performance benefits. In the following sections, we turn to probabilistic SPJ queries.

6.5.3 Probabilistic Selection: prob-sel vs. sel-ML

In this experiment, we compare the answer quality and performance of the probabilistic selection prob-sel and the optimized selection over the ML world sel-ML.opt. We use a set of selection conditions on each of the five labels in the Contact dataset: *companyname*, *firstname*, *lastname*, *jobtitle* and *department*. For each label we use all the values for that label in the ground truth to construct the set of selection conditions, and measure the average false positive and false negative rates. This way, we avoid the randomness due to a particular selection condition.

The table in Figure 6.12 shows the false negative and false positive rates for the two approaches over the five selection condition sets. As can be seen in the table, the prob-sel algorithm achieves much lower false negatives. For example, the false negatives for *department* is reduced to 30%. The results also show that prob-sel maintains a false positive rate $\leqslant 0.01$ for all the cases.

In many IE application scenarios false negatives are more problematic than false positives because the former means losing answers, and the latter means extra answers, which can be remedied by further processing. Our goal is to reduce the false negative rate without significantly increasing the false positive rate.

Note that in the results shown here, The probability `threshold` in prob-sel algorithm was set to 0.001—the estimated lowest probability of the ML extractions based on a sample documents. Increasing this `threshold` would exclude some ML extractions, which would then increase the false negatives. Decreasing the `threshold` has minimal effect on the false negatives, while increasing the false positive rate slightly. For the queries used in this experiment, all the false positive rates remain below 0.05 even when the `threshold` is set as low as 10^{-6}.

The right most bars for each selection condition in Figure 6.9 shows the execution time of prob-sel algorithm, compared to sel-ML.naive and sel-ML.opt over $100k$ documents from DBLP. It can be seen that while significantly reducing the false negative rates, the execution time of the prob-sel algorithm is 2 to 5 times that of the corresponding sel-ML.opt algorithm. The is the trade-off between the answer quality and execution time.

(False -)	k=1	k=5	k=10	k=15	k=20
join-ML	0.373	0.373	0.373	0.373	0.373
probjoin.naive	0.373	0.253	0.245	0.216	0.191
probjoin.opt	0.373	0.312	0.283	0.260	0.251
(False +)	k=1	k=5	k=10	k=15	k=20
join-ML	0.082	0.082	0.082	0.082	0.082
probjoin.naive	0.082	0.259	0.437	0.515	0.567
probjoin.opt	0.082	0.091	0.093	0.097	0.110

Figure 6.13. False negative, false positive rates (Contact) between join-ML.opt and prob-join algorithms.

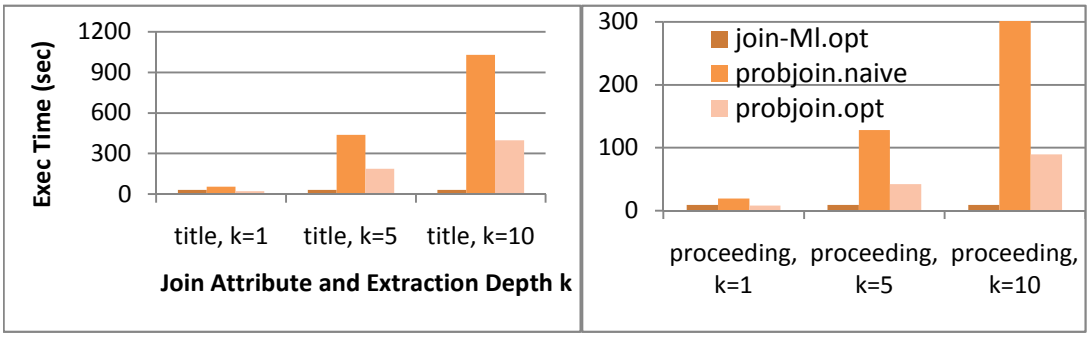


Figure 6.14. Performance comparison (DBLP) between join-ML.opt and prob-join algorithms.

We argue that a probabilistic IE system should support both algorithms, so that the right choice can be made based on the requirements of the application.

6.5.4 Probabilistic Join: prob-join vs. join-ML

Finally, we describe a set of experiments comparing the answer quality and performance of the probabilistic join (prob-join) and the optimized join over the ML world (join-ML.opt). Two prob-join algorithms are prob-join.naive, which joins over top-k worlds, and prob-join.opt, which performs an incremental join as described in Section 6.4.3.

The experiments for answer quality are conducted on the Contact dataset, and the query is to find all Contact pairs with the same *companyname*. The table in Figure 6.13 shows that, for both prob-join.opt and prob-join.naive, the false negative rate decreases, and the false positive rate increases as we increase the maximum extraction depth k. prob-join.opt achieves a substantial (slightly over 33%) reduction in false negatives when $k = 20$, with only a moderate increase of the false positive rate. On the other hand, prob-join.naive, although achieving a more significant reduction of the false negative rate, incurs 5 times increase in the false positive rate compared to join-ML.opt. Thus, for most purposes, join-ML.opt achieves the best answer quality among the three algorithms.

The performance experiments are conducted for join queries on *title* and *proceeding* attributes between two document sets with $10k$ and $1k$ documents respectively randomly picked from the DBLP dataset. As shown in Figure 6.14, for the two join queries with varying settings of k, the join-ML.opt performs significantly better than both prob-join.naive and prob-join.opt when $k > 1$. The optimized algorithm prob-join.opt with the incremental join achieves a significant 2 to 3 times speedup compared to the naive algorithm prob-join.naive. The execution time of both prob-join.naive and prob-join.opt grows linearly with k. Thus, k provides a tuning knob that can be used to adjust the accuracy/performance tradeoff of the prob-join approach.

To summarize, the experiments reported in Section 6.5.1 and 6.5.2 demonstrated that integrating the Viterbi inference and relational operators can significantly improve performance of SPJ-over-ML queries without loss of answer quality. The results in Section 6.5.3 and 6.5.4 showed that the answer quality (i.e. false negatives) can be improved significantly by probabilistic SPJ queries. The answer quality comes in some cases (e.g., probabilistic selection) at a moderate cost, and in others (e.g., probabilistic join) at a higher cost. One advantage of representing the IE model and inference algorithms in database is that a cost-based query optimizer can choose between the different execution techniques explored in this chapter.

6.6 Summary

Common industry practice, described in Section 1.4.3 loosely connects statistical IE packages and database systems to perform probabilistic data analysis over unstructured data. Such an approach suffers from both performance and accuracy limitations. In this chapter, we proposed two families of queries over CRF-based IE models in the BAYESSTORE setting. The first query family includes deterministic queries over maximum-likelihood extractions. We also developed optimizations to push the relational operators into the Viterbi algorithm. The second query family extends the Viterbi algorithm to produce a set of possible extraction "worlds", from which we compute top-k probabilistic query answers. As the experimental results show, this approach can improve the runtime efficiency of the first query family and improve the accuracy, especially recall, using the second query family. These two query families also provide a design space in which answer quality and performance can be traded off according to the needs of the application.

Chapter 7

Query Processing and Optimization with MCMC Inference

In this chapter, we describe the probabilistic query processing algorithms and optimization techniques enabled by the different inference algorithms described in Chapter 5, including the general MCMC inference. Using these inference algorithms, we are able to extend the limited class of top-k style probabilistic queries based on Viterbi, described in the previous chapter, to a wider range of queries involving marginal inference, aggregation, and non-linear-chain CRF models. With multiple in-database inference algorithms, a query can now be executed using different query plans, involving different inference algorithms. This opens up the opportunity for query optimization.

First, we describe the template of the extended set of queries that can be supported by BAYESSTORE. We use examples to justify the need for general inference algorithms, such as MCMC algorithms. We develop techniques to achieve a deep integration between the MCMC algorithms with the relational operators. We compare the characteristics of different inference algorithms over CRF models both qualitatively and quantitatively. We describe the *hybrid inference* technique, a database rewrite algorithm, which can choose the proper inference algorithm for different subsets of the data for a single probabilistic query. The experimental results show significant speed-ups using the hybrid inference optimization. These results demonstrate the potential to use declarative languages and a query optimizer to speed up probabilistic queries that involve both relational and inference operators.

7.1 Overview

In Chapters 3 to 6, we have described important components in BAYESSTORE to support probabilistic declarative IE applications, including the in-database representation of CRF models, the in-database implementation of the Viterbi algorithm, and the probabilistic SPJ queries over the probabilistic extracted entities. However, the techniques described so far are limited by the capa-

bilities of the Viterbi algorithm, which can only handle top-k-style queries over a limited class of CRF models—linear chain CRFs.

In order to deal with non-linear CRF models such as skip-chain CRF models to handle repeated terms in the text [12], complex IE queries that induce cyclic models over the linear-chain CRFs, and marginal inference queries that produce richer probabilistic outputs than top-k inference, inference algorithms other than Viterbi are needed. In this chapter, we explore the broader problem of effectively supporting general probabilistic inference inside a PDB-based declarative IE system.

In the following sections, we first describe the template of the extended set of queries that can be supported by BAYESSTORE. We use examples to justify the need for general inference algorithms, such as MCMC algorithms. We also develop query-driven techniques to achieve a deep integration between the in-database MCMC implementation, described in Chapter 5.4.1, with the relational operators.

We compare the applicability of the four inference algorithms, including two variations of the general sampling-based Markov chain Monte Carlo (MCMC) inference algorithm—Gibbs sampling and MCMC Metropolis-Hastings (MCMC-MH)—in addition to the Viterbi and the sum-product algorithms. We study the data and the model parameters that affect the accuracy and runtime of those algorithms. Based on those parameters, we develop a set of rules for choosing an inference algorithm based on the characteristics of the model and the data.

We study the integration of relational query processing and statistical inference algorithms, and demonstrate that, for SQL queries over probabilistic extraction results, the proper choice of IE inference algorithm is not only model-dependent, but is also query- and text-dependent. Such dependencies arise when relational queries are applied to CRF models, inducing additional variables, edges, and cycles, and when the model is instantiated over different text data, resulting in model instances with drastically different characteristics.

To achieve good accuracy and runtime performance, it is imperative for a PDB (probabilistic database) system to use a hybrid approach to IE even within a single query, employing different algorithms for different records. In the context of our CRF-based PDB system, we describe query processing steps and an algorithm to generate query plans that apply hybrid inference for probabilistic IE queries.

Finally, we describe example queries and experimental results showing that such hybrid inference techniques can improve the runtime of the query processing by taking advantage of the appropriate inference methods for different combinations of query, text, and CRF model parameters. We evaluate our approaches and algorithms using three real-life datasets: DBLP, NYTimes, and Twitter. The results show that our hybrid inference techniques can achieve up to 10-fold speedups compared to the non-hybrid standard solutions practiced in industry.

7.2 Query Template

The queries we consider are probabilistic queries, which inference over the probabilistic attribute `label`^{p} in the TOKENTBL table. Each TOKENTBL is associated with a specific CRF model stored in the MR table. Such CRF-based IE is captured by a sub-query with logic that produces the IE

```
SELECT  Top-k(T1.docID,[T1.pos|exist]) |
        [Marginal(T1.docID,[T1.pos|exist])] |
        [Top-k(T1.docID, T2.docID,[T1.pos|T2.pos|exist])] |
        [Marginal(T1.docID,T2.docID,[T1.pos|T2.pos|exist])]
FROM    TokenTbl1 T1[, TokenTbl2 T2] WHERE   T1.label = 'bar1' [and
T1.token = 'foo1']
        [and T1.docID = X] [and T1.pos = Y]
        [and T1.label = T2.label] [and T1.token = T2.token]
        [and T1.docID = T2.docID]
GROUP BY T1.docID[, T2.docID] HAVING  [aggregate condition]
```

Figure 7.1. The SQL+IE query template.

results from the base probabilistic TOKENTBL tables. The sub-query consists of a relational part Q_{re} over the probabilistic token tables TOKENTBL and the underlying CRF models, followed by an inference operator Q_{inf}. A canonical "query template" captures the logic for the SQL+IE sub-query in Figure 7.1. The query template supports SPJ queries, aggregate conditions, and two types of inference operators, `Top-k` and `Marginal`, over the probabilistic TOKENTBL tables.

The relational part of a SQL+IE query Q_{re} first specifies, in the `FROM` clause, the TOKENTBL table(s) over which the query and extraction is performed.

The `WHERE` clause lists a number of possible selection as well as join conditions over the TOKENTBL tables. These conditions when involve $\texttt{label}^p$ are probabilistic conditions, and deterministic otherwise. For example, a probabilistic condition `label='person'` specifies that the entity type we are looking for is `'person'`, while a deterministic condition `token='Bill'` specifies that the name of the entity we are looking for is `'Bill'`. We can also specify a join condition `T1.token=T2.token and T1.label=T2.label` indicating that two documents need to contain the same entity name with the same entity type.

In the `GROUP BY` and the `HAVING` clause in the query template, we can specify conditions on an entire text "document". An example of such an aggregate condition over a bibliography document is that all `title` tokens are in front of all the `author` tokens. Following the possible world semantics, described in Section 2.4.1, the execution of these relational operators involve modification to the original graphical models as will be shown in Section 7.3.

The inference part Q_{inf}, of a SQL+IE query, takes the $docID$, the pos, and the CRF model resulting from Q_{re} as input. The inference operation is specified in the `SELECT` clause, which can be either a `Top-k` or a `Marginal` inference. The inference can be computed over different random variables in the CRF model: (1) a sequence of tokens (e.g., a document) specified by `docID`; or (2) a token at a specific location specified by `docID` and `pos`; or (3) the "existence" (`exist`) of the result tuple. The "existence" of the result tuple becomes probabilistic with a selection or join over a probabilistic attribute, where `exist` variables are added to the model as described in Section 4.

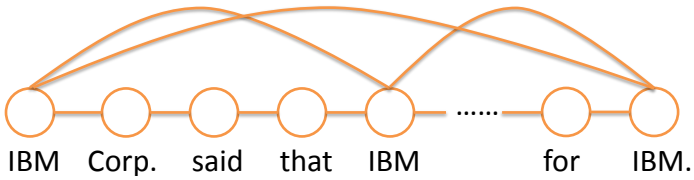


Figure 7.2. A skip-chain CRF model that includes skip-edges between non-consecutive tokens with the same string (e.g.,"IBM").

For example, the inference `Marginal(T1.docID,T1.pos)`, for each position `(T1.docID,T1.pos)` computed from Q_{re}, returns the distribution of the `label` variable at that position. The inference `Marginal(T1.docID,exist)` computes the marginal distribution of `exist` variable for each result tuple. We can also specify an inference following a join query. For example, the inference `Top-k(T1.docID,T2.docID)`, for each document pair `(T1.docID,T2.docID)`, returns the `top-k` highest probability joint extractions that satisfy the join constraint.

7.3 Cycles from IE Models and Queries

In many IE tasks, good accuracy can only be achieved using non-linear CRF models like skip-chain CRF models, which model the correlation not only between the labels of two consecutive tokens as in linear-chain CRF, but also between those of non-consecutive tokens. For example, a correlation can be modeled between the labels of two tokens in a sentence that have the same string. Such a skip-chain CRF model can be seen in Figure 7.2, where the correlation between non-consecutive labels (i.e., skip-chain edges) form cycles in the CRF model.

In simple probabilistic databases with independent base tuples, a class of "safe plans" introduced in [26] gives rise to tree-structured graphical models [29], where the exact inference is tractable. However, in a CRF-based IE setting, an inverted-file representation of text in TOKENTBL inherently has cross-tuple correlations. Thus, even queries with "safe plans" over simple linear-chain CRF models, result in cyclic models and intractable inference problems.

For example, the following query computes the marginal inference `Marginal(T1.docID, T2.docID,exist)`, which returns pairs of `docIDs` and the probabilities of the existence (`exist`) of their join results. The join query is performed between each document pair on having the same token strings labeled as 'person'. The join query over the base TOKENTBL tables adds cross-edges to the pair of linear-chain CRF models underlying each document pair. Figure 7.3(a),(b) shows two examples of the resulting CRF model after the join query over two different pairs of documents. As we can see, the CRF model in (a) is tree-shaped, and the one in (b) is cyclic.

```
Q1: [Probabilistic Join Marginal]
SELECT  Marginal(T1.docID,T2.docID,exist)
FROM    TokenTbl1 T1, TokenTbl2 T2
WHERE   T1.label = T2.label and T1.token = T2.token
            and T1.label = 'person';
```

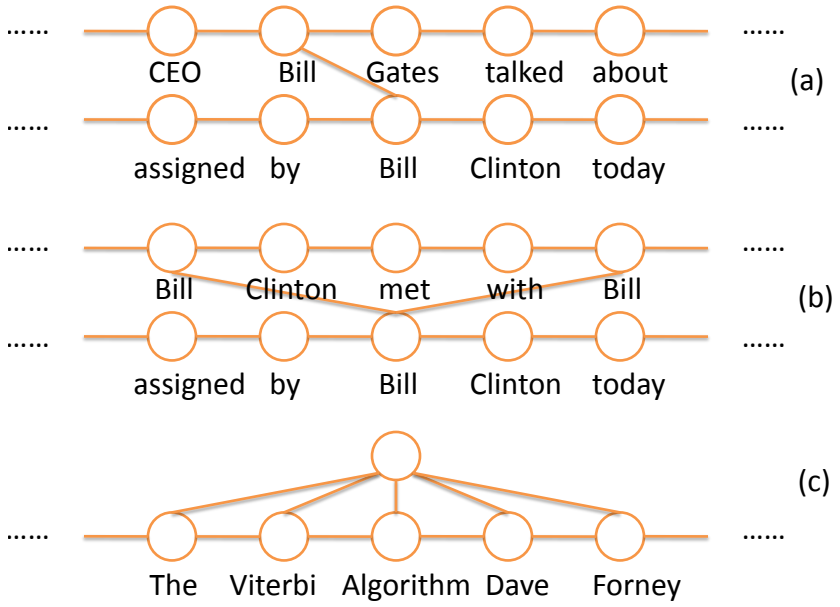


Figure 7.3. (a) and (b) are example CRF models after applying a join query over two different pairs of documents. (c) is the resulting CRF model from a query with an aggregate condition.

Another example is a simple query to compute the top-k extraction conditioned on an aggregate constraint over the label sequence of each document (e.g., all "title" tokens are in front of "author" tokens). This query induces a cyclic model as shown in Figure 7.3(c).

```
Q2: [Aggregate Constraint Top-k]
SELECT  Top-k(T1.docID)
FROM    TokenTbl1 T1
GROUP   BY docID
HAVING  [aggregate constraint] = true;
```

7.4 Query-Driven MCMC Sampling

In Section 5.4, we described two general MCMC inference algorithms for such cyclic models. Moreover, in Section 6.3, we developed query-driven techniques to integrate probabilistic selection and join conditions into the Viterbi algorithm over linear-chain CRF models. However, the kinds of constraints that Viterbi can handle are limited and specific to the Viterbi.

In this section, we explore query-driven techniques for the sampling-based MCMC inference algorithms. Query-driven sampling is needed to compute inference conditioned on the query constraints. Such query constraints can be highly selective, where most samples generated by the basic MCMC methods do not "qualify" (i.e., satisfy the constraints). Thus, we need to adapt the MCMC methods by pushing the query constraints into the sampling process. Note that our adapted, query-driven MCMC methods still converge to the target distribution as long as the proposal function can reach every "qualified" world that satisfies the constraints in a finite number of steps.

There are three types of query constraints: (1) selection constraints; (2) join constraints; and (3) aggregate constraints. Both (1) and (2) were studied for Viterbi in the last chapter. The following query contains an example selection constraint, which is to find the top-k highest likelihood extraction that contains a 'person' entity 'Bill'.

```
Q3: [Selection Constraint Top-k]
SELECT  Top-k(T1.docID)
FROM    TokenTbl1 T1
WHERE   token = 'Bill' and label = 'person'
```

An example of a join constraint can be found in $Q1$ in Section 7.3, and an example of an aggregate constraint can be found in $Q2$ in the same Section.

The naive way to answer those conditional queries using *MCMC* methods is to: first, generate a set of samples using Gibbs sampling or MCMC-MH regardless of the query constraint; second, filter out the samples that do not satisfy the query constraint; last, compute the query over the remaining qualified samples.

In contrast, our query-driven sampling approach pushes the query constraints into the MCMC sampling process by restricting the worlds generated by GENINITWORLD(), GENPROPOSALS() and GENSAMPLES() functions, so that all the samples generated satisfy the query constraint. One of the advantages of the MCMC algorithms is that the proposal and sample generation functions can naturally deal with the deterministic constraints, which might induce cliques with high tree-width in the graphical model. Such cliques can easily "blow up" the complexity of known inference algorithms [10]. We exploit this property of MCMC to develop query-driven sampling techniques for different types of queries.

The query-driven GENINITWORLD() function generates an initial world that satisfies the constraint. There are two possible ways to generate the first qualified sample:

- **Special Values:** It turns out that, for most query constraints, we can directly set the variable values to those specified in the query. For `Q3`, it sets the labels for all 'Bill' tokens to 'person' in each document. For `Q1`, it sets the labels for a pair of common tokens to 'person' in each document pair. To satisfy the aggregate constraint in `Q2`, it sets the labels for all tokens to be 'title'.
- **Random Sample:** In the general case, we can fall back to generating random samples until one that satisfies the constraint is found.

The query-driven GENPROPOSAL() and GENSAMPLES() functions are called iteratively to generate new samples that satisfy the constraint. There are two possible ways to generate the next qualified jump (i.e., new sample):

- **Restricted Jumps:** One way is to apply restrictions on which labels can be assigned to the variables from which to draw the next sample. For `Q3`, it requires that the label for at least one 'Bill' token in each document is 'person'. For `Q1`, it requires that the labels for at least one pair of common tokens in each document pair are 'person'.

inference algorithm	Top-k Chain	Tree	Cyclic	Marginal Chain	Tree	Cyclic	Constraints Some	Arbitrary
Viterbi	✓						✓	
sum-product				✓	✓		✓	
MCMC-Gibbs	✓	✓	✓	✓	✓	✓	✓	
MCMC-MH	✓	✓	✓	✓	✓	✓		✓

Table 7.1. Applicability of different inference algorithms for different queries (e.g., top-k, marginal) over different model structures (e.g., linear-chain, tree-shaped, cyclic), and in handling query constraints.

- **Random Jumps:** In the general case, we can generate samples according to the Gibbs or the MCMC-MH algorithm, and reject ones which do not satisfy the query constraint. We apply this rejection-based technique for aggregate constraints like the one in `Q2`.

In some cases, the query constraints divide the space of the conditional distribution into multiple disconnected regions, which cannot be reached from one to the other within finite single jumps (i.e., changing the value of a single variable). In those cases, MCMC Metropolis-Hastings can be adapted to take multiple jumps, changing values of multiple variables in one sample, while Gibbs sampling cannot be. One example constraint that needs proposal functions with multiple jumps is the join constraint in `Q1`. This is one advantage of the MCMC-MH algorithm over Gibbs, i.e., that it handles arbitrary query constraints.

7.5 Choosing Inference Algorithms

Different inference algorithms over probabilistic graphical models have been developed in a diverse range of communities (e.g., natural language processing, machine learning, statistical physics, etc). The characteristics of these inference algorithms (e.g., applicability, accuracy, convergence rate, runtimes) over different model structures have since been studied to help modeling experts select an appropriate inference algorithm for a specific problem [10].

In this section, we first compare the characteristics of the four inference algorithms we have developed over CRF models. Next we introduce parameters that capture important properties of the model and data. Using these parameters, we then describe a set of rules to choose among different inference algorithms.

7.5.1 Comparison between Inference Algorithms

We have implemented four inference algorithms over CRF models for IE applications: (1) Viterbi, (2) sum-product, and two sampling-based MCMC methods: (3) Gibbs Sampling and (4) MCMC Metropolis-Hastings (MCMC-MH). In Table 7.1, we show the applicability of these algorithms to different inference tasks (e.g. top-k, or marginal) on models with different structures (e.g., linear-chain, tree-shaped, cyclic).

As we can see, Viterbi, Gibbs and MCMC-MH can all compute top-k queries over linear-chain

CRF models; sum-product, Gibbs and MCMC-MH can all compute marginal queries over linear-chain and tree-shaped models; while only MCMC algorithms can compute queries over cyclic models. Although there are heuristic adaptations of the sum-product algorithm for cyclic models, past literature found MCMC methods to be more effective in handling complicated cyclic models with long-distance dependencies and deterministic constraints [74, 75]. In terms of handling query constraints, the Viterbi and sum-product algorithms can only handle selection constraints, Gibbs sampling can handle selection constraints and aggregate constraints that do not break the distribution into disconnected regions. On the other hand, MCMC-MH can handle arbitrary constraints in SQL+IE queries.

7.5.2 Parameters

Next, we introduce a list of parameters that affect the applicability, accuracy and runtime of the four inference algorithms that we have just described:

1. **Data Size:** the size of the data is measured by the total number of tokens in information extraction;
2. **Structure of Grounded Models:** the structural properties of the model instantiations over data:
   (a) shape of the model (i.e., linear-chain, tree-shaped, cyclic),
   (b) maximum size of the clique,
   (c) maximum length of the loops (e.g., skip-chain in linear CRF)
3. **Correlation Strength:** the relative strength of transitional correlation between different label variables;
4. **Label Space:** the number of possible labels.

The data size affects the runtime for all the inference algorithms. The runtime of the Viterbi and sum-product algorithms is linear in the data size. The MCMC algorithms are iterative optimizations that can be stopped at any time, but the number of samples needed to converge depends linearly on the size of the data.

The structure of the grounded model can be quantified with three parameters: the shape of the model, the maximum size of the clique, and the maximum length of the loops. The first parameter determines the applicability of the models, and is also the most important factor in the accuracy and the runtime of the inference algorithms over the model. In addition, the maximum clique size and the length of the loops play an important role in the runtime of several known inference algorithms (including, for example, the junction tree and the loopy belief propagation algorithms) [10]. In this chapter, we ignore the latter two parameters.

The correlation strength is the relative strength of the transition correlation between different label variables over the state correlation between tokens with their corresponding labels. The correlation strength does not influence the accuracy or the runtime of the Viterbi or the sum-product

algorithm. However, it is a significant factor in the accuracy and runtime of the MCMC methods, especially the Gibbs algorithm. Weaker correlation strengths result in faster convergence for the Gibbs sampler. At the extreme, zero transition correlation results in complete label independence, rendering consecutive Gibbs samples independently, which would converge very quickly.

The size of the label space of the model is also an important factor of the runtime of all the inference algorithms. The runtime of the Viterbi and sum-product algorithms is quadratic in the size of the label space, while the runtime of the Gibbs algorithm is linear in the label space because each sampling step requires enumerating all possible labels.

7.5.3 Rules for Choosing Inference Algorithms

Among the parameters described in the previous section, we focus on (1) the shape of the model, (2) the correlation strength, and (3) the label space, because the rest are less influential in the four inference algorithms we study in this chapter. The data size is important for optimizing the extraction order in the join over top-k queries as described in [76]. However, since the complexity of the inference algorithms we study in this chapter are all linear in the size of the data, it is not an important factor for choosing inference algorithms.

Based on the analysis in the previous section, we use the following rules to choose an inference algorithm for different data and model characteristics, quantified by the three parameters, and the query:

- For cyclic models:
  - If cycles are induced by query constraints, choose query-driven MCMC-MH over Gibbs Sampling;
  - Otherwise, choose Gibbs Sampling over MCMC-MH. As shown in our experiments in Sections 7.7.2-7.7.3, the Gibbs Sampler converges much faster than the random walk MCMC-MH for computing both top-k extractions and marginal distribution;
- For acyclic models:
  - For models with small label space, choose Viterbi over MCMC methods for top-k and sum-product over MCMC methods for marginal queries;
  - For models with strong correlations, choose Viterbi and sum-product over MCMC methods;
  - For models with both a large label space and weak correlations, choose Gibbs Sampling over MCMC-MH, Viterbi, and sum-product.

For a typical IE application, the label space is small (e.g., 10 to 20), and the correlation strength is fairly strong. For example, `title` tokens are usually followed by the `author` tokens in a bibliography string. Moreover, strong correlation exists with any multi-token entity names (e.g., a `person` token is likely to be followed by another `person` token). Thus, we believe that the above rules translate in most cases in IE to: choose Viterbi and sum-product over MCMC methods for

acyclic models for top-k and marginal queries respectively; choose Gibbs Sampling for cyclic models unless the cycles are induced by query constraints, in which case choose query-driven MCMC-MH. In this thesis, we use heuristic rules to decide the threshold for a "small" label space and for a "strong" correlation for a data set.

7.6 Hybrid Inference

Typically, for a given model and dataset, a single inference algorithm is chosen based on the characteristics of the model. In this section, we first show that in the context of SQL queries over probabilistic IE results, the proper choice of IE inference algorithm is not only model-dependent, but also query- and text-dependent. Thus, to achieve good accuracy and runtime performance, it is imperative for a PDB system to use a hybrid approach to IE even within a single query, employing different inference algorithms for different records.

We describe the query processing steps that employ hybrid inference for different documents within a single query. Then we describe an algorithm, which, given the input of a SQL+IE query, generates a query plan that applies the hybrid inference. Finally, we show the query plans with hybrid inference generated from three example SQL+IE queries to take advantage of the appropriate IE inference algorithms for different combinations of query, text and CRF models.

7.6.1 Query Processing Steps

In the context of SQL queries over probabilistic IE results, the proper choice of the IE inference algorithm depends on many factors. In addition to the probabilistic model, this choice also depends on the query and the text as we explain below.

First, the relational sub-query Q_{re} augments the original model with additional random variables, cross-edges and factor tables, making the model structure more complex, as we explained in Section 7.3. The characteristics of the model may change after applying the query over the model. For example, a linear-chain CRF model may become a cyclic CRF model, as happens with the join query in `Q1` and the query with aggregate constraint in `Q2`.

Second, when the resulting CRF model is instantiated (i.e., grounded) over a document, it could result in a grounded CRF model with drastically different model characteristics. For example, the CRF model resulting from a join query over a linear-chain CRF model, when instantiated over different documents can result in either a cyclic or a tree-shaped model, as we have shown in Figure 7.3(a) and (b).

The applicability, accuracy and runtime of different inference algorithms vary significantly over models with different characteristics, which can result from different data for the same query and model. As a result, to achieve good accuracy and runtime, we apply different inference algorithms (i.e., hybrid inference) for different documents within a single query. The choice of the inference algorithm over a document is based on the characteristics of its grounded model, and rules for choosing inference algorithms we described in Section 7.5.3.

The main steps in query processing with hybrid inference are as follows:

1. **Apply Query over Model:** Apply the relational part of the query Q_{re} over the underlying CRF model;

2. **Instantiate Model over Data:** Instantiate the resulting model from the previous step over the text, and compute the important characteristics of the grounded models;

3. **Partition Data:** Partition the data according to the properties of grounded models from the previous step. In this thesis, we only partition the data according to the shape of the grounded model. More complicated partitioning techniques, such as one based on the size of the maximum clique can be considered for future work;

4. **Choose Inference:** Choose the inference algorithms to apply according to the rules in Section 7.5.3 over the different data partitions based on the characteristics of the grounded models;

5. **Execute Inference:** Execute the chosen inference algorithm over each data partition, and return the union of the results from all partitions.

7.6.2 Query Plan Generation Algorithm

We envision that the query parser takes in a SQL+IE query and outputs, along with other non-hybrid plans, a query plan which applies hybrid inference over different documents.

The algorithm in Figure 7.4 generates a hybrid inference query plan for an input SQL+IE query Q, consisting of the relation part Q_{re}, and the subsequent inference operator Q_{inf}. In Line 1, the relational operators in Q_{re} are applied to the CRF models underlying the base TOKENTBL tables, resulting in a new CRF model CRF^*. In Lines 2 to 3, selection and join conditions on the deterministic attributes (e.g., docID, pos, token) are applied to the base TOKENTBL tables, resulting in a set of tuples T, each of which represents a document or a document pair. In Line 4, the model instantiation is applied over T using CRF^* to generate a set of "ground" models `groundCRFs`. In Line 5, a split operation is performed to partition the `groundCRFs` according to their model structures into `linearCRFs`, `treeCRFs` and `cyclicCRFs`.

Lines 6 to 19 capture the rules for choosing inference algorithms we described in Section 7.5.3. Finally, a union is applied over the result sets from different inference algorithms for the same query.

Lines 11 to 14 deals with a special set of queries, which compute the top-k results over a simple query with aggregate conditions that induce cycles over the base linear-chain CRFs. The intuition is that it is always beneficial to apply the Viterbi algorithm over the base linear-chain CRFs as a fast filtering step before applying the MCMC methods. In Line 12, it first computes the top-k extractions `res` without the aggregate constraint using Viterbi. In Line 13, it applies the constraint to the top-k extractions in `res`, which results in a set of top-k extractions that satisfy the constraints in `res1`. In Line 14, the query-driven MCMC-MH is applied to the documents in T with extractions that do not satisfy the constraint: `(res-res1).T`. An example of this special case is described in Section 7.6.3.

HYBRID-INFERENCE-PLAN-GENERATOR (Q)

apply Q_{re} *over the base CRF models* $\rightarrow$ *CRF**
apply deterministic selections in Q *over base* TOKENTBL*s* $\rightarrow \{T_i\}$
apply deterministic joins in Q *over* $\{T_i\} \rightarrow T$
apply model instantiation over T *using* $CRF^* \rightarrow$ *groundCRFs*
apply split operation to groundCRFs $\rightarrow$ *linearCRFs*, *treeCRFs*, *cyclicCRFs*
if Q_{inf} *is Marginal* **then**
  apply sum-product to $(linearCRFs + treeCRFs) \rightarrow$ *res2*
  apply Gibbs to $(cyclicCRFs) \rightarrow$ *res3*
else if Q_{inf} *is Top-*k **then**
  apply Viterbi to $(linearCRFs) \rightarrow$ *res1*
  if Q_{re} *contains aggregate constraint but no join* **then**
    apply Viterbi to $(cyclicCRFs + treeCRFs) \rightarrow$ *res*
    apply aggregate constraint in Q *over res* $\rightarrow$ *res1*
    apply query-driven MCMC-MH to $(res - res1).T \rightarrow$ *res3*
  else
    apply Gibbs to $(cyclicCRFs + treeCRFs) \rightarrow$ *res3*
endif endif
if Q_{inf} *is Top-*k **then**
  apply union of res1 and res3
else if Q_{inf} *is Marginal* **then**
  apply union of res2 and res3
endif

Figure 7.4. Pseudo-code for the hybrid inference query plan generation algorithm.

Complexity: The complexity of generating the hybrid plan depends on the complexity of the operation on Line 5 in Figure 7.4, where the `groundCRFs` are split into subsets of `linearCRFs`, `treeCRFs` and `cyclicCRFs`. The split is performed by traversing the ground CRFs to determine their structural properties, which is linear to the size of the ground CRF $O(N)$, where N is the number of random variables. The complexity of choosing the appropriate inference (lines 6 to 21) is $O(1)$.

On the other hand, the complexity of Viterbi, sum-product algorithms over `linearCRFs` and `treeCRFs` is $O(N)$ with a much larger constant, because of the complex computation (i.e., sum, product) over $|Y| \times |Y|$ matrices, where $|Y|$ is the number of possible labels. The complexity of the exact inference algorithm over `cyclicCRFs` is NP-hard. Thus the cost of generating the hybrid plan is negligible from the cost of the inference algorithms.

7.6.3 Example Query Plans

In this section, we describe three example queries and show the query plans with hybrid inference generated from the algorithm in Figure 7.4, which take advantage of the appropriate inference algorithms for different combinations of query, text and CRF models.

Skip-Chain CRF

In this query, we want to compute the marginal distribution or the top-k extraction over an underlying skip-chain CRF model as shown in Figure 7.2. The query is simply:

```
Q4: [Skip-Chain Model]
SELECT  [Top-k(T1.docID) | Marginal(T1.pos|exist)]
FROM    TokenTbl T1
```

As described in [68], the MCMC methods are normally used to perform inference over skip-chain CRF models for all the documents, like the query plan in Figure 7.5(b). The Viterbi algorithm fails to apply because a skip-chain CRF model contains skip-edges, which make the CRF model cyclic.

However, the existence of such skip-edges in the grounded models, instantiated from the documents, is dependent on the text! There exist documents, like the one shown in Figure 7.2, in which one string appears multiple times. Those documents result in cyclic CRF models. But, there also exist documents, in which only unique tokens are used except for "stop-words", such as "for", "a". Those documents result in linear-chain CRF models.

The query plan generated with hybrid inference is shown in Figure 7.5(a). After the model instantiation, the ground CRF model is inspected: if no skip-edge exists (i.e., no duplicate strings exist in a document), then the Viterbi or the sum-product algorithm is applied; otherwise, the Gibbs algorithm is applied to the cyclic ground CRFs. Compared to the non-hybrid query plan, the query plan with hybrid inference is more efficient by applying more efficient inference algorithms (e.g., Viterbi, sum-product) over the subset of the documents, where the skip-chain CRF model does not induce cyclic graphs. The speedup depends on the performance of Viterbi/sum-product compared

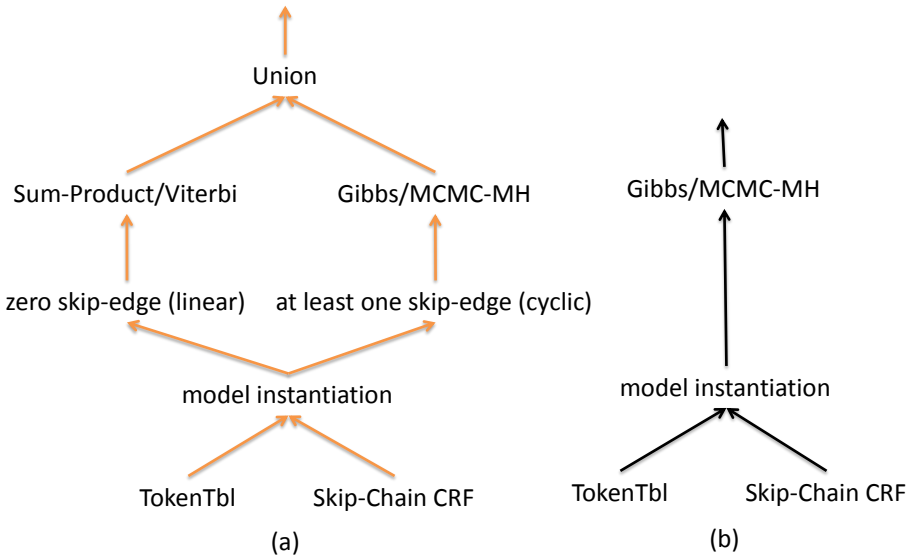


Figure 7.5. The query plan for an inference, either top-k or marginal, over a skip-chain CRF model.

to Gibbs Sampling, and on the percentage of such documents that instantiate a skip-chain CRF model into a grounded linear-chain CRF models.

Join over Linear-chain CRF

In this example, we use the join query `Q1` described in Section 7.3, which computes the marginal probability of the existence of a join result. The join query is performed between each document pair on having tokens with the same strings labeled as 'person'.

Such a join query over the underlying linear-chain CRF models induces cross-edges and cycles in the resulting CRF model. A typical non-hybrid query plan, shown in Figure 7.6 with black edges, perform MCMC inference over all the documents.

However, as we see in Figure 7.3(a) and (b), depending on the text, the joint CRF model can be instantiated into either a cyclic graph or a tree-shaped graph. The red edge in Figure 7.6 shows the query plan with hybrid inference for the join query `Q1`. As we can see, instead of performing MCMC methods unilaterally across all "joinable" document pairs (i.e., contain at least 1 pair of common tokens), the sum-product algorithm is used over the document pairs that contain only 1 pair of common tokens. Compared to the non-hybrid query plan, the hybrid inference reduces the runtime by applying the more efficient inference (i.e., sum-product) when possible. The speedup depends on the performance of sum-product compared to the MCMC methods, and the percentage of the "joinable" document pairs that only share one pair of common tokens that are not "stop-words".

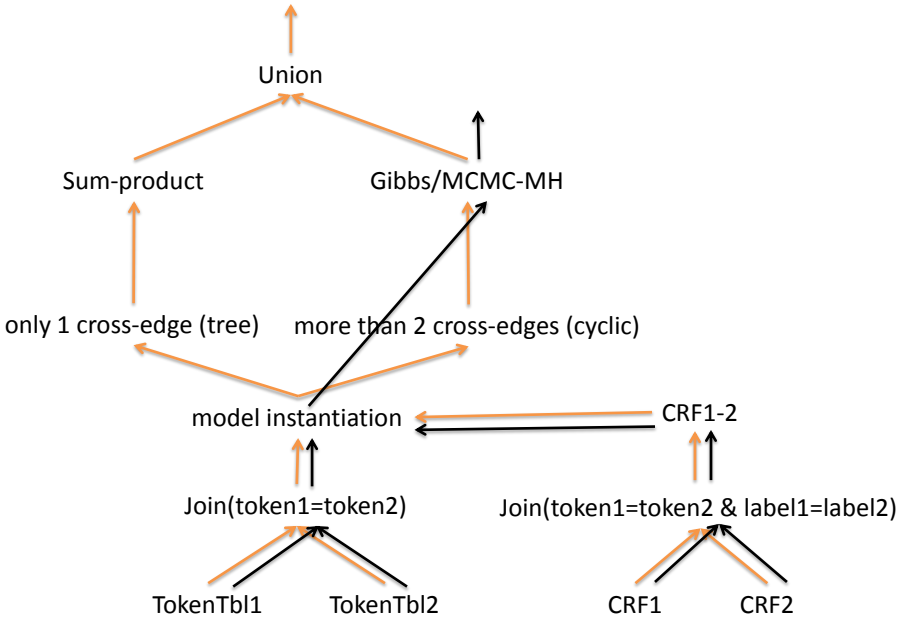


Figure 7.6. The query plan for the probabilistic join query followed by marginal inference.

Aggregate Constraints

In this example, we use `Q2`, the query with an aggregate constraint, described in Section 7.3. As shown in Figure 7.3(c), the aggregate constraints can induce a big clique including all the label variables in each document. In other words, regardless of the text, based on the model and the query, each document is instantiated into a cyclic graph with high tree-width.

Again, typically, for such a high tree-width cyclic model, MCMC-MH algorithms are used over all the documents to compute the top-k extractions that satisfy the constraint. Such a non-hybrid query plan is shown in Figure 7.7(b).

However, this query falls into the special case described in the query plan generation algorithm in Section 7.6.2 for hybrid inference. The query is to return the top-k extractions over the cyclic graph induced by an aggregate constraint over a linear-chain CRF model. Thus, the resulting query plan is shown in Figure 7.7(a).

In the query plan with hybrid inference, the Viterbi algorithm runs first to compute the top-k extraction without the constraint. Then, the results are run through the aggregate: those that satisfy the constraint are returned as part of the results, and those that do not satisfy the constraint are fed into the query-driven MCMC-MH algorithm.

7.7 Experimental Results

So far, we have described the query-driven optimization of the MCMC algorithms, and the query plans for the hybrid inference algorithms. We now present the results of a set of experiments aimed

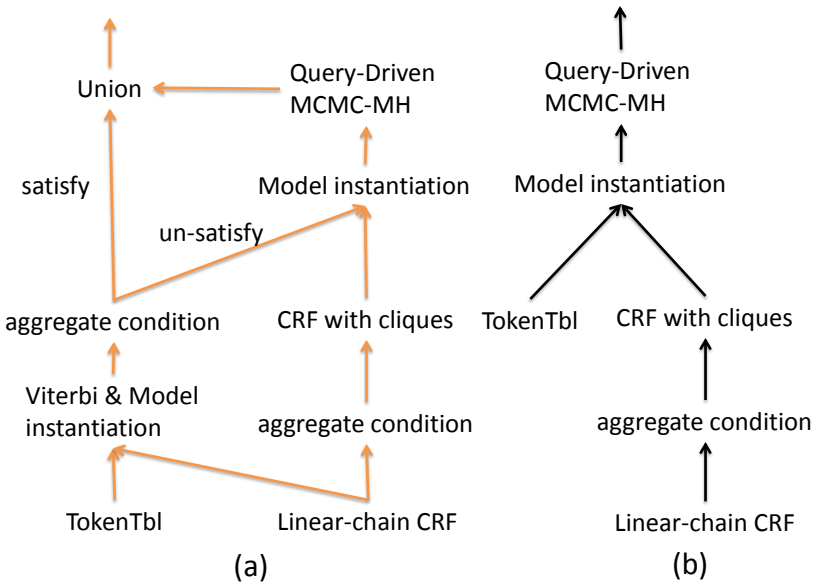


Figure 7.7. The query plan for the aggregate selection query followed by a top-k inference.

to (1) evaluate the effectiveness of the query-driven sampling techniques in improving the runtime of the MCMC algorithms; (2) compare the accuracy and runtime of the four inference algorithms: Viterbi, sum-product, Gibbs and MCMC-MH, for the two IE tasks—top-k and marginal; and (3) analyze three real-life text datasets to quantify the potential speedup of a query plan with hybrid inference compared to one with non-hybrid inference.

Setup and Dataset: We implemented the four inference algorithms: Viterbi, sum-product, Gibbs and MCMC-MH in PostgreSQL 8.4.1. We conducted the experiments reported here on a 2.4 GHz Intel Pentium 4 Linux system with 1GB RAM.

For evaluating the effectiveness of the query-driven optimizations of the MCMC methods, and for comparing the accuracy and runtime of the inference algorithms, we use the DBLP dataset [69] and a CRF model with 10 labels and features similar to those in [20]. The DBLP database contains more than $700,000$ papers with attributes, such as conference, year, etc. We generate bibliography strings from DBLP by concatenating all the attribute values of each paper record.

For quantifying the speedup of query plans with hybrid inference, we examine the New York Times (NYTimes) dataset, and the Twitter dataset in addition to the DBLP dataset. The NYTimes dataset contains ten-million tokens from $1,788$ New York Times articles from the year 2004. The Twitter dataset contains around $200,000$ tokens from over $40,000$ tweets obtained in January 2010. We label both corpora with 9 labels, including person, location, etc.

7.7.1 Query-Driven MCMC-MH

In this experiment, we evaluate the effectiveness of the query-driven MCMC-MH algorithm described in Section 7.4 with the basic MCMC-MH in generating samples that satisfy the query

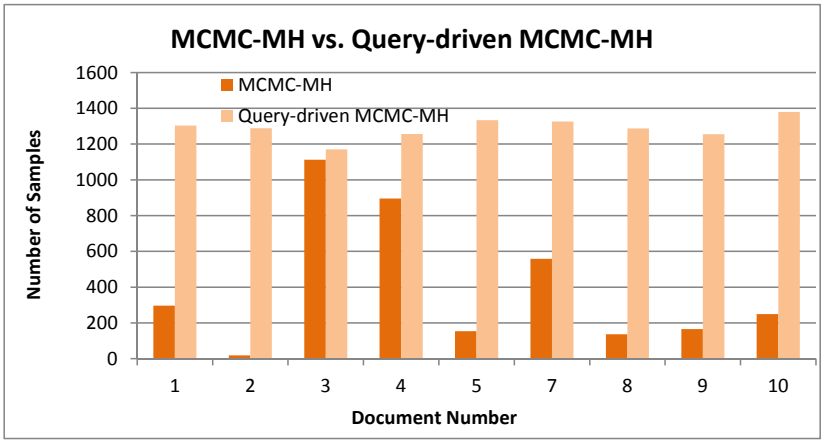


Figure 7.8. The number of qualified samples generated by the query-driven MCMC-MH and the basic MCMC-MH algorithm in 1 second for different documents in DBLP.

constraint. The query we use is `Q2` described in Section 7.3, which computes the top-1 extractions that satisfy the aggregate constraint that all `title` tokens are in front of the `author` tokens.

We run the query-driven MCMC-MH and the basic MCMC-MH algorithm over a randomly picked 10 documents from the DBLP dataset. Figure 7.8 shows the number of qualified samples that are generated by each algorithm in 1 second. As we can see, the query-driven MCMC-MH generates more qualified samples, roughly 1200 for all the documents, and for half of the documents the query-driven MCMC-MH generates more than 10 times more qualified samples than basic MCMC-MH.

7.7.2 MCMC vs. Viterbi on Top-k Inference

This experiment compares the runtime and the accuracy of the Gibbs, the MCMC-MH and the Viterbi algorithms in computing top-1 inference over linear-chain CRF models. The inference is performed over $45,000$ tokens in 1000 bibliography strings from the DBLP dataset. We measure the "computation error" as the number of labels different from the exact top-1 labelings according to the model [1]. The Viterbi algorithm only takes 6.1 seconds to complete the exact inference over these documents, achieving zero computation error.

For the MCMC algorithms, we measure the computation error and runtime for every $10k$ more samples, starting from $10k$ to 1 million samples over all documents. As we can see in Figure 7.9, the computation error of the Gibbs algorithm drops to 22% from $45,000$ to $10,000$ when $500k$ samples are generated. This takes around 75 seconds, more than 12 times longer than the runtime of the Viterbi algorithm. The MCMC-MH converges much slower than the Gibbs Sampling. As more samples are generated, the top-1 extractions generated from the MCMC algorithms get closer and closer to the exact top-1, however very slowly. Thus, Viterbi beats the MCMC methods by

[1]The top-1 extractions with zero computation error may still contain mistakes, which are caused by inaccurate models.

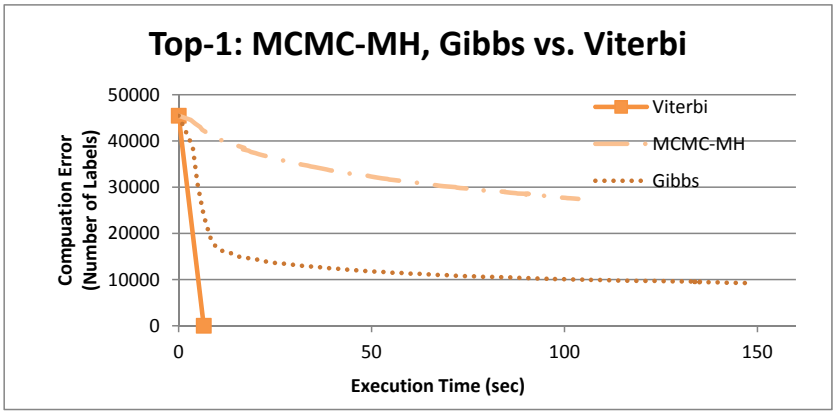


Figure 7.9. Runtime-Accuracy graph comparing Gibbs, MCMC-MH and Viterbi over linear-chain CRFs for top-1 inference on DBLP.

far in computing top-1 extractions with linear-chain CRF models: more than 10 times faster with more than 20% fewer computation errors.

7.7.3 MCMC vs. Sum-Product on Marginal Inference

This experiment compares the runtime and the accuracy of the Gibbs, MCMC-MH and the sum-product algorithms over tree-shaped graphical models induced by a join query similar to `Q1`, described in Section 7.3. The query computes the marginal probability of the existence of a join result for each document pair in DBLP, joining on the same 'publisher'. The query is performed over a set of $10,000$ pairs of documents from DBLP, where the two documents in each pair have exactly one token in common.

The sum-product algorithm over these $10,000$ tree-shaped graphical models takes about 60 seconds. As an exact algorithm, the sum-product algorithm achieves zero computation error. We measure the "computation error" as the difference between the marginal probabilities of join computed from the MCMC-MH algorithms and the sum-product algorithm, averaging over all document pairs.

For the MCMC algorithms, we measure the computation error and runtime for every $200k$ more samples, starting from $200k$ to 2 million samples over all document pairs. As we can see in Figure 7.10, the probability difference between Gibbs and sum-product converges to zero quickly: at 400 second, the probability difference is dropped to 0.01. The MCMC-MH on the other hand, converges much slower than the Gibbs.

This experiment shows that the MCMC algorithms perform relatively better in computing marginal distributions than in computing top-1 extractions. However, the sum-product algorithm still out-performs the MCMC algorithms in computing marginal probabilities over tree-shaped models: more than 6 times faster with about 1% less computation error.

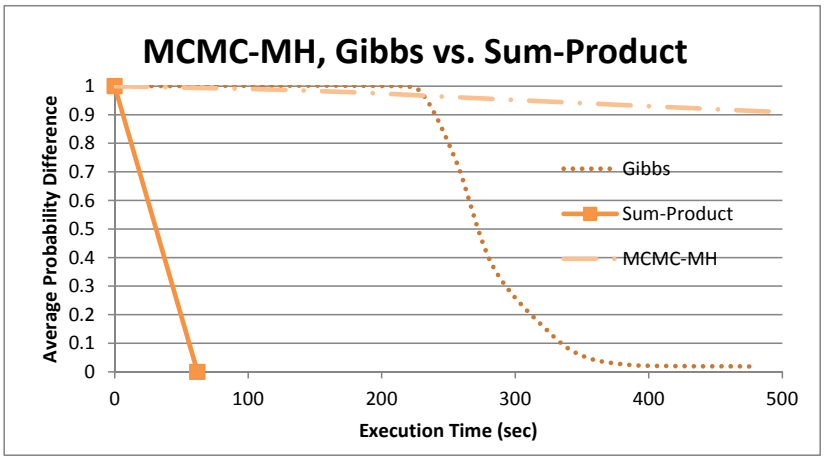


Figure 7.10. Runtime-Accuracy graph comparing Gibbs, MCMC-MH and sum-product over tree-shaped models for marginal inference on DBLP.

7.7.4 Exploring Model Parameters

In this experiment, we explore how different correlation strengths, one of the parameters we discussed in Section 7.7.4, affect the runtime and the accuracy of the inference algorithms. As we explained earlier, the correlation strength does not affect the accuracy or the runtime of the Viterbi algorithm. On the other hand, weaker correlation between different random variables in the CRF model leads to faster convergence for the MCMC algorithms. The setup of this experiment is the same as in Section 7.7.2.

In Figure 7.11, we show the runtime-accuracy graph of the Viterbi and the Gibbs algorithm to compute the top-1 extractions over models with different correlation strengths. We synthetically generated models with correlation strengths of 1, 0.5, 0.2 and 0.001 by dividing the original scores in the transition factors by 1, 2, 5 and 1000 respectively. As we can see, the weaker correlation strengths lead to faster convergence for the Gibbs algorithm. When correlation strength is 0.001 the computation error reduces to zero in less than twice that of the Viterbi runtime.

The correlation strength of the CRF model depends on the dataset on which the CRF model is learned. The model we learned over NYTimes and DBLP dataset both contains strong correlation strength.

7.7.5 Hybrid Inference for Skip-chain CRFs

In this and the next two sections, we describe the results, in terms of runtime speed-ups, comparing the query plan generated by hybrid inference with a non-hybrid solution. Table 7.2 summarizes the results of hybrid inference for different queries over different models.

The query we use in this experiment is `Q4` over the skip-chain CRF model, described in Section 7.6.3. Given that the Viterbi is more than 10 times more efficient than Gibbs with zero computation error, as we showed in Section 7.7.2, the speed-up enabled by the hybrid inference for `Q4`

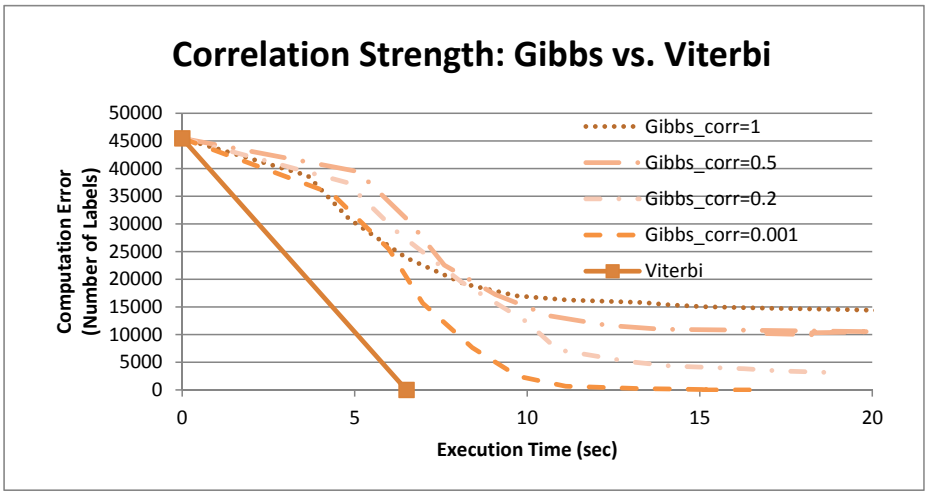


Figure 7.11. Runtime-Accuracy graph comparing Gibbs and Viterbi over models with different correlation strengths on DBLP.

Data Corpora	Skip-chain CRF	Probabilistic Join	Aggregate Constraint
NYTimes	$\times 5.0$	$\times 4.5$	$\times 10.0$
Twitter	$\times 5.0$	$\times 2.6$	N/A
DBLP	$\times 1.0$	$\times 1.0$	N/A

Table 7.2. Speed-ups achieved by hybrid inference for different queries.

is determined by the percentage of the documents that do not contain duplicate non-"stop-word" tokens.

For NYTimes dataset, we use the sentence breaking function in NLTK toolkit [77], and the full-text stop-word list from MySQL [78]. Over all sentences, only about 10.3% contain duplicate non-"stop-word" tokens. Thus, the optimizer will use the Viterbi algorithm for 89.7% of the sentences, while using the Gibbs algorithm for the rest. This hybrid inference plan can achieve a 5-fold speedup compared to the non-hybrid solution, where the Gibbs algorithm is used over all the documents.

We did the same analysis on the Twitter dataset. The number sentences that contains non-"stop-word" duplicate tokens is 10.0%, which leads to a similar 5-fold speedup. On the other hand, for DBLP dataset, the number of documents that contains non-"stop-word" duplicates is as high as 96.9%, leading to a 3% speedup.

7.7.6 Hybrid Inference for Probabilistic Join

The query used in this experiment is the join query `Q1`, described in Section 7.6.3. Given that the sum-product is more than 6 times more efficient than Gibbs with zero computation error, as we showed in Section 7.7.3, the speed-up enabled by the hybrid inference for `Q1` is determined by the percentage of the "joinable" document pairs that share only one pair of common non-"stop-word" tokens.

For the NYTimes dataset, about 6.0% "joinable" sentence pairs share more than one pair of non-"stop-word" common tokens. Thus, the sum-product algorithm can be applied to the other 93.6% of the sentences, achieving a 4.5 times speedup compared to the non-hybrid approach of running Gibbs over the joint CRF model of all the document pairs.

For the Twitter datset, around 25.6% "joinable" sentence pairs share more than one pair of tokens. This is much lower than NYTimes dataset mainly because tweets contain a lot of common shorthands. Thus the speedup is around 2.6 times. For DBLP dataset, on the other hand, the speedup is little due to the high percentage of document pairs contain more than one pair of common words.

7.7.7 Hybrid Inference for Aggregate Constraint

The query used in this experiment is `Q2` with an aggregate constraint described in Section 7.6.3. We performed this query over the DBLP dataset. Out of all the top-1 extractions of the $10,000$ bibliography strings, only 25 of them do not satisfy the aggregate constraint that all `title` tokens are in front of all `author` tokens. Thus, although the aggregate constraint in the query induce a big clique in the CRF model, which calls for MCMC algorithms, the MCMC is not needed for most of the cases. To perform `Q2` over DBLP, MCMC only needs to be performed over 25 out of $10,000$ documents, which leads to a 10-fold speedup.

The results in Section 7.7.1 show that the query-driven sampling techniques can effectively generate more samples that satisfy query constraints for conditional queries. Section 7.7.2 and Section 7.7.3 show that the Viterbi and sum-product algorithms are by far more efficient and more

accurate than the MCMC algorithms over linear-chain and tree-shaped models in IE. Lastly, based on the text analysis over NYTimes, Twitter and DBLP datasets, we conclude that the query plans with hybrid inference can achieve up to a 10-fold speed-up compared to the non-hybrid solutions in cases where the MCMC inference algorithms are only applied over a very small percentage of the data, whereas Viterbi or Sum-Product is applied over the majority of the data.

7.8 Summary

In this chapter, we show that general inference algorithms, such as MCMC inference, are needed to execute probabilistic queries beyond the top-k-style queries, involving marginal inference, cyclic models, and aggregation operators. Using the in-database implementation of the MCMC algorithms, we show that such native implementations enable efficient probabilistic query processing with a tight integration of the inference and relational operators. The hybrid inference optimization demonstrates the feasibility and potential of using a query optimizer to support a declarative query language for different inference operations over probabilistic graphical models. Results from three real-life datasets demonstrate that the hybrid inference can achieve up to a 10-fold speed-up compared to the non-hybrid solutions in cases where the MCMC inference algorithms are only applied over a very small percentage of the data, whereas Viterbi or Sum-Product is applied over the majority of the data.

Chapter 8

Future Work

In this Chapter, we discuss directions of future research related to query-driven machine learning systems and extensile framework and interface design for probabilistic databases.

Query-Driven Machine Learning System Based on our experience in BAYESSTORE, we envision a *query-driven machine learning* (QDML) system, which can (1) integrate ML methods with query processing; (b) optimize ML methods given user queries; (3) incorporate user feedback to improve the ML models and improve the inference results; and (4) compose ad-hoc complex queries that involve multiple ML methods and relational query operators. The BAYESSTORE system took the first steps to explore the possibilities and benefits of a QDML system in two application domains. However, this QDML approach can enable many other novel and useful functionalities for ML-based data analysis, which has not been explored in BAYESSTORE to date.

First, learning algorithms need to be implemented in databases (BAYESSTORE only supports inference algorithms). We are especially interested in the class of learning algorithms called online learning algorithms, which can modify the model as new training data comes in without reconstructing the whole model from the scratch. Combining model learning functions with relational operators, we can specify any subset of the data on which the model is to be learned. Moreover, we can build a set of models using the `GROUP BY` operator – one for each group. Finally, we can define model factors as views over the data, which enables the model to be updated as soon as the data is updated.

Second, query-driven inference techniques need to be developed for ML models and inference algorithms other than CRFs and Viterbi, so that these inference algorithms can be computed efficiently over the models at query-time. Such query-driven ML system also enables users to compose complex queries over the output of multiple ML models and algorithms.

Third, cost-based optimizer needs to be developed to optimize various statistical models and methods. For example, given a text analysis task, there are different models and tools with different accuracy and runtime tradeoffs. We start to explore this tradeoff for the different inference algorithms over linear-chain CRF models in the BAYESSTORE work. However, the optimization we came up with results in a heuristic optimizer for a single ML model and task. We see the need

of a cost-based optimizer to optimize multiple statistical models and methods. Furthermore, with the extension to the in-database learning operators, the optimizer should also be able to pick which learning algorithm and even which statistical model to use to best answer a user query.

Extensible Framework: As we have described in Section 2.1, probabilistic graphical models are a class of important and widely used statistical models for probabilistic data analysis in real-life applications. The BAYESSTORE project has been focused on how to efficiently support two types of PGM's (i.e., FOBN and CRF) in two BAYESSTORE sub-systems. These two BAYESSTORE systems both represent the uncertain data as incomplete relations and the model as factor tables. However, we implemented different inference and relational algorithms in these two sub-systems for different PGM models and methods.

There are many other different types of PGM's such as naive Bayes, hidden Markov models, logistic regression, and Markov networks. Each type of model has different characteristics, which are tuned towards different types of applications. Moreover, more models are being developed to solve new problems. Thus, to develop an in-database model representation and inference algorithms in a BAYESSTORE-like system for each particular model from scratch is very time consuming. What we need is an extensible architecture in which new models can be plugged into this architecture by specifying a set of model-specific functions.

Future work is needed to come up with this set of functions, as well as to design a framework with an effective development environment to plug in a new PGM model and its methods. This framework should also be able to support scalable deployment of these models and methods, as well as efficient optimizations between them and the relational operators. Another very important research direction is to come up with user interfaces and visualization tools for non-IT people (i.e., those without ML, DB, or Map-Reduce knowledge) who are experts in their application domains (e.g., environmentalists, doctors, social scientists) to be able to interact with their data, easily apply different ML methods, and to effectively give feedback to the results of data analysis.

Chapter 9

Concluding Remarks

In this thesis, we proposed, built, and evaluated BAYESSTORE, a probabilistic database system that natively supports SML models and various inference algorithms to perform advanced data analysis. This marriage of database and SML technologies creates a declarative and efficient probabilistic processing framework for applications dealing with large-scale uncertain data. We explored a variety of research challenges, including extending the database data model with probabilistic data and statistical models, defining relational operators (e.g., select, project, join) over probabilistic data and models, developing joint optimization of inference operators and the relational algebra, and devising novel query execution plans. We used sensor networks data analysis and information extraction over text as the two driving applications.

The main contributions of this thesis include: (1) we designed an efficient in-database representation for uncertain data and for probabilistic models; (2) we invented new algorithms for probabilistic relational operators, including select, project, and join over probabilistic models and data; (3) we developed native in-database implementations for four inference algorithms over probabilistic graphical models, including Viterbi, sum-product, MCMC Metropolis Hastings, and MCMC Gibbs sampling; (4) we crated query-driven techniques for executing PDA queries that involve both inference operators and relational operators; and (5) we developed a query optimization technique called "hybrid inference", which can choose among the different inference algorithms we implemented in BAYESSTORE.

This thesis work shows that: First, sophisticated statistical methods, involving recursive and iterative procedures, can be efficiently implemented in a set-oriented data processing framework like PostgreSQL and achieve comparable runtime to open source implementation of the same algorithms in procedural languages; Second, querying over distributions rather than the top-1 highest-probability results can reduce false negatives (i.e., missing results) by as much as 80%; Third, the query-driven inference techniques can achieve orders-of-magnitude speed-up on large-scale datasets compared to the batch process used in the common practice of loose coupling for PDA; Lastly, query optimization techniques for inference queries can achieve significant speed-ups compared to standard algorithms used in the SML literature.

The thesis not only shows the design and implementation of a probabilistic database system

based on PGMs for probabilistic data analysis, but also opens up the opportunity of deeper integration between database systems and a wide range of ML methods to perform more advanced and scalable data analysis. Moving forward, we believe it is important to further develop the query-driven inference techniques to a broader class of ML models and algorithms in order to build a query-driven machine learning system that can support efficient and scalable ad-hoc data analysis queries. We also see the need to develop an extensible framework that can support various statistical ML models and their inference and learning algorithms.

Bibliography

[1] "Extracting Value from Chaos," in IDC Digital Universe Study, sponsored by EMC., June 2011.

[2] B. Warneke, M. Last, B. Liebowitz, and K. Pister, "Smart dust: Communicating with a cubic-millimeter," in Computer, 2001.

[3] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," in *A*SPLOS, 2000.

[4] S. Madden, M. Franklin, and J. Hellerstein, The Design and Evaluation of a Query Processing Architecture for Sensor Networks. PhD thesis, University of California, Berkeley, 2003.

[5] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in Proceedings of the 30th VLDB Conference, (Toronto, Canada), 2004.

[6] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling," in Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics. http://nlp.stanford.edu/software/CRF-NER.shtml, 2005.

[7] A. McCallum and A. Kachites, "MALLET: A Machine Learning for Language Toolkit," in http://mallet.cs.umass.edu, 2002.

[8] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu, "Mystiq: a system for finding more answers by using probabilities," in Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp. 891–893, 2005.

[9] O. Benjelloun, A. Sarma, A. Halevy, and J. Widom, "ULDB: Databases with Uncertainty and Lineage," in *V*LDB, 2006.

[10] D. Koller and N. Friedman, "Probabilistic Graphical Models: Principles and Techniques," 2009.

[11] M. I. Jordan, Learning in graphical models. Springer, 2004.

[12] L. Getoor and B. Taskar, Introduction to Statistical Relational Learning. The MIT Press, 2007.

[13] C. M. Bishop, "Pattern Recognition and Machine Learning," in Springer, 2007.

[14] S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach," in Prentice Hall, 2009.

[15] A. Darwiche, "Modeling and Reasoning with Bayesian Networks," in Cambridge University Press, 2009.

[16] M. Jordan, "Graphical models," in *S*tatistical Science (Special Issue on Bayesian Statistics), 19:140–155, 2004.

[17] C. Sutton and A. McCallum, "Introduction to Conditional Random Fields for Relational Learning," in *Introduction to Statistical Relational Learning*, 2008.

[18] J. Lafferty, A. McCallum, and F. Pereira, "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data," in *ICML*, 2001.

[19] G. D. Forney, "The Viterbi Algorithm," IEEE, vol. 61, pp. 268–278, March 1973.

[20] T. Kristjansson, A. Culotta, P. Viola, and A. McCallum, "Interactive Information Extraction with Constrained Conditional Random Fields," in AAAI'04, 2004.

[21] K. P. Murphy, Y. Weiss, and M. I. Jordan, "Loopy belief propagation for approximate inference: An empirical study," in In Proceedings of Uncertainty in AI, 1999, 1999.

[22] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: An efficient data clustering method for very large databases," in Proc. SIGMOD Conference, 1996.

[23] J. C. Shafer, R. Agrawal, and M. Mehta, "Sprint: A scalable parallel classifier for data mining," in Proc. VLDB Conference, 1996.

[24] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in Proc. VLDB Conference, 1994.

[25] D. Suciu, D. Olteanu, C. R, and C. Koch, Probabilistic Databases. 2011.

[26] N. Dalvi and D. Suciu, "Efficient Query Evaluation on Probabilistic Databases," in *V*LDB, 2004.

[27] N. Dalvi and D. Suciu, "Management of Probabilistic Data: Foundations and challenges," in *P*ODS, 2007.

[28] R. Gupta and S. Sarawagi, "Curating Probabilistic Databases from Information Extraction Models," in *V*LDB, 2006.

[29] P. Sen and A. Deshpande, "Representing and Querying Correlated Tuples in Probabilistic Databases," in *I*CDE, 2007.

[30] P. Sen, A. Deshpande, and L. Getoor, "Exploiting shared correlations in probabilistic databases," PVLDB, vol. 1, no. 1, pp. 809–820, 2008.

[31] T. Imieliński and W. Lipski, "Incomplete Information in Relational Databases," in JACM, 31(4), 1984.

[32] R. Cavallo and M. Pittarelli, "The Theory of Probabilistic Databases," in VLDB, 1987.

[33] D. Barbará, H. Garcia-Molina, and D. Porter, "The Management of Probabilistic Data," IEEE Trans. on Knowl. and Data Eng., vol. 4, no. 5, pp. 487–502, 1992.

[34] N. Fuhr and T. Rolleke, "A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems," ACM Transactions on Information Systems, vol. 15, pp. 32–66, 1997.

[35] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian, "Probview: a flexible probabilistic database system," in ACM Trans. Database Syst., p. 22:419469, 1997.

[36] R. Cheng, S. Singh, and S. Prabhakar, "U-dbms: a database system for managing constantly-evolving data," in VLDB '05: Proceedings of the 31st international conference on Very large data bases, pp. 1271–1274, VLDB Endowment, 2005.

[37] A. Deshpande and S. Madden, "MauveDB: Supporting Model-based User Views in Database Systems," in SIGMOD, 2006.

[38] D. Wang, E. Michelakis, M. Garofalakis, and J. Hellerstein, "BayesStore: Managing Large, Uncertain Data Repositories with Probabilistic Graphical Models," in VLDB, 2008.

[39] L. Antova, T. Jansen, C. Koch, and D. Olteanu, "Fast and Simple Relational Processing of Uncertain Data," in *ICDE*, 2008.

[40] R. Jampani, L. Perez, M. Wu, F. Xu, C. Jermaine, and P. Haas, "MCDB: A Monte Carlo Approach to Managing Uncertain Data," in SIGMOD, 2008.

[41] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. Hambrusch, and R. Shah, "Orion 2.0: Native Support for Uncertain Data," in Proceeding of the ACM Special Interest Group on Management of Data (SIGMOD 2008), 2008.

[42] R. Cheng, S. Singh, S. Prabhakar, R. Shah, J. S. Vitter, and Y. Xia, "Efficient join processing over uncertain data," in CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management, (New York, NY, USA), pp. 738–747, ACM Press, 2006.

[43] P. Sen, A. Deshpande, , and L. Getoor, "Representing tuple and attribute uncertainty in probabilistic databases," in DUNE, 2007.

[44] F. Xu, K. S. Beyer, V. Ercegovac, P. J. Haas, and E. J. Shekita, "E = mc^3: managing uncertain enterprise data in a cluster-computing environment," in SIGMOD Conference, pp. 441–454, 2009.

[45] P. J. Haas, C. M. Jermaine, S. Arumugam, F. Xu, L. L. Perez, and R. Jampani, "Mcdb-r: Risk analysis in the database," PVLDB, vol. 3, no. 1, pp. 782–793, 2010.

[46] L. Antova, C. Koch, and D. Olteanu, "World-set decompositions: Expressiveness and efficient algorithms," in Proceedings of ICDT, 2007.

[47] R. Ramakrishnan and J. Gehrke, Database Management Systems. McGraw-Hill Higher Education, 2000.

[48] M. Kamber and J. W. Han, "Data Mining: Concepts and Techniques, Third Edition," in The Morgan Kaufmann Series in Data Management Systems, 2011.

[49] H. Bravo and R. Ramakrishnan, "Optimizing MPF Queries: Decision Supposrt and Probabilistic Inference," in Proceedings of SIGMOD '07, 2007.

[50] F. Niu, C. Re, A. Doan, and J. Shavlik, "Tuffy: Scaling up statistical inference in markov logic networks using an rdbms," in Proceedings of VLDB, 2011.

[51] H. Poon and P. Domingos, "Joint inference in information extraction," in Proceedings of AAAI, 2007.

[52] P. Singla and P. Domingos, "Entity resolution with markov logic," in Proceedings of ICDE, 2006.

[53] S. Riedel and I. Meza-Ruiz, "Collective semantic role labeling with markov logic," in Proceedings of CoNLL, 2008.

[54] F. Wu and D. Weld, "Automatically refining the wikipedia infobox ontology," in Proceedings of WWW, 2008.

[55] U. of Washington, "Alchemy,"

[56] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer, "Learning Probabilistic Relational Models," in *IJCAI*, 1999.

[57] A. Pfeffer, "*Probabilistic Reasoning for Complex Systems*," in PhD thesis, Stanford, 2001.

[58] D. Poole, "First-order probabilistic inference," in Proceedings of the 2003 International Joint Conferene on Artificial Intelligence (IJCAI), 2003.

[59] L. Getoor, Learning Statistical Models from Relational Data. PhD thesis, Stanford University, 2002.

[60] L. Getoor, D. Koller, and N. Friedman, "From instances to classes in probabilistic relational models," in Proceedings of the ICML Workshop on Attribute-Value and Relational Learning: Crossing the Boundaries, 2000.

[61] M. Richardson and P. Domingos, "Mln: A declarative language for implementing dynamic programs," in Machine Learning, 2006.

[62] J. Eisner, E. Goldlust, and N. A. Smith, "Compiling Comp Ling: Weighted Dynamic Programming and the Dyna Language," in Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT-EMNLP), pp. 281-290, 2005.

[63] J. Eisner, E. Goldlust, and N. A. Smith, "Dyna: A Declarative Language for Implementing Dynamic Programs," in Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL), Companion Volume, pages 218-221, 2004.

[64] B. Taskar, P. Abbeel, and D. Koller, "Discriminative Probabilistic Models for Relational Data," in *U*AI, 2002.

[65] E. Michelakis, D. Wang, M. Garofalakis, and J. Hellerstein, "Granularity conscious modeling for probabilistic databases," in *D*UNE, 2007.

[66] S. Sarawagi, "CRF Open Source Project. http://crf.sourceforge.net/,"

[67] R. Thomason in http://www.eecs.umich.edu/r̃thomaso/bibs/bigbibs.html.

[68] M. Wick, A. McCallum, and G. Miklau, "Scalable Probabilistic Databases with Factor Graphs and MCMC," in VLDB2010, 2010.

[69] "DBLP dataset," in http://kdl.cs.umass.edu/data/dblp/dblp-info.html.

[70] "Enron email dataset," in http://www.cs.cmu.edu/ enron/.

[71] L. Huang and D. Chiang, "Better k-best Parsing," in IWPT, 2005.

[72] I. F. Ilyas, R. Shah, W. G. Aref, J. S. Vitter, and A. K. Elmagarmid, "Rank-aware Query Optimization," in SIGMOD, 2004.

[73] "Contact record extraction data," in http://www2.selu.edu/Academics/Faculty/aculotta/data/contact.html.

[74] B. Milch, B. Marthi, and S. Russell, BLOG: Relational Modeling with Unknown Objects. PhD thesis, University of California, Berkeley, 2006.

[75] A. McCallum, K. Schultz, and S. Singh, "Factorie:probabilistic programming via imperatively defined factor graphs," in NIPS, 2009.

[76] D. Wang, M. Franklin, M. Garofalakis, and J. Hellerstein, "Querying Probabilistic Information Extraction," in PVLDB, 2010.

[77] "NLTK toolkit," in http://www.nltk.org/.

[78] "MySQL full-text stop-words," in http://dev.mysql.com/doc/refman/5.1/en/fulltext-stopwords.html.

Appendix A

Pseudo-codes

A.1 In-database Implementation of MCMC-MH

A.2 Probabilistic Projection followed by Top-k Inference over CRF

```
1  CREATE FUNCTION MCMC-MH (int) RETURN VOID AS
2  $$
3  -- compute the initial world: getInitialWorld()
4  insert into MHSamples
5  select setval('world_id',1) as worldId, docId, pos, token,
           trunc(random()*num_label+1) as label
6  from tokentbl;

7  -- compute N sample proposals: genProposals()
8  insert into Proposals
9  with X as (
     select id, docID, trunc(random()*docLen(docID)) as pos
     from (
10     select generate_series as id,
         ((generate_series-1)/($1/numDoc)+1) as docID
11     from generate_series(1,$1)) foo )
12 select X.id,S.docId,S.pos,S.token,
     trunc(random()*numLabel+1) as label,
     null::integer[] as prevWorld, null::integer[] as factors
13 from X, tokentbl S
14 where X.docID = S.docID and X.pos = S.pos;

15 -- fetch context: initial world and factor tables
16 update proposals S1
17   set prev_world = (select * from getInitialWorld(S1.docId))
18   from proposals S2
19   where S1.docId <> S2.docId and S1.id = S2.id+1;
20 update proposals S1
21   set factors = (select * from getFactors(S1.docId))
22   from proposals S2
23   where S1.docId <> S2.docId and S1.id = S2.id+1;

24 -- accept/reject proposals: getAlpha()
25 insert into MHSamples
26 select worldId, docId, pos, token, label
27 from (
28     select nextval('world_id') worldId, docId, pos, token, label,
                getalpha_agg((docId,pos,label,0.0,prev_world,factors)
                ::getalpha_io) over (order by id) alpha
29     from (select * from proposals order by id) foo) foo
30 where (foo.alpha).docID <> -1;
31 $$
32 LANGUAGE SQL;
```

Figure A.1. The SQL implementation of MCMC Metropolis-Hasting takes input N – the total number of samples to generate in MHSAMPLES table.

TOP1PROBPROJECT (doc, χ)

1 $V(1, y_1)[1].score \leftarrow \text{MR}(doc.x_1, y_1, -1); V(1, y_1)[1].prev = \{-1, -1\};$

2 **for** *each token* $doc.x_i (i = 2, ..., T)$ **do**

3 **for** *each cur label* $y \in Y$ **do**

4 GET-NEXT-U(V, i, y, χ); GET-NEXT-V(V, i, y, χ);

5 **endfor endfor**

6 $Tmp \leftarrow \cup_{y_T \in Y} V(T, y_T); Tmp \leftarrow Tmp \cup$ GET-NEXT-U$(V, T + 1, -1, \chi)$;

7 *sort* Tmp *by score desc*; $y^*[T] \leftarrow Tmp[1]$;

8 **for** $i = T - 1; i >= 0; i - -$ **do**

9 $label \leftarrow y^*[i + 1].prev.label$;

10 **if** $label =$ 'don't care' **then** $y^*[i] \leftarrow U(i, y^*[i + 1].label)[1]$;

11 **else** $y^*[i] \leftarrow V(i, label)[1]$;

12 **endif endfor**

13 **return** y^*

GET-NEXT-V (V, i, y, χ)

1 $maxidx \leftarrow V(i, y).length$;

2 **if** $maxidx > 0$ **then**

3 $py \leftarrow V(i, y).prev.label; pidx \leftarrow V(i, y).prev.idx$;

4 **if** $py =$ 'don't care'&&$pidx = U(i, y).length$ **then** GET-NEXT-U(V, i, y, χ);

5 **else if** $py \neq$ 'don't care'&&$pidx = V(i - 1, py).length$ **then** GET-NEXT-V$(V, i - 1, py, \chi)$;

6 **endif endif**

7 $Tmp \leftarrow \cup_{y' \in Y \setminus \chi} V(i - 1, y')$;

8 **for** *each* $te = \{score, label, prev\} \in Tmp$ **do**

9 $te.score \leftarrow te.score + \text{MR}(x_i, te.label, y')$;

10 **endfor**

11 $Tmp \leftarrow Tmp \cup U(i, y)$; *sort* Tmp *by score desc*;

12 $V(i, y) \leftarrow V(i, y) \cup Tmp[maxidx + 1]$;

Figure A.2. Algorithm for computing top-1 result for probabilistic projection queries.

GET-NEXT-U (V, i, y, χ)

1 $maxidx \leftarrow U(i,y).length; labels[i] \leftarrow \chi;$

2 **for** $l = 1; true; l++$ **do**

3 $upper \leftarrow \Sigma_{y \in \chi} V(i,y').$GET-LAST$().score;$

4 *// UT is the rank-join buffer*

5 $score \leftarrow UT(i,y)[1].score;$

6 **if** $upper \leqslant score$ **then**

7 $U(i,y)[maxidx+1] \leftarrow \{score, y, UT(i,y)[1].prev\};$

8 $UT(i,y) \leftarrow UT(i,y) - UT(i,y)[1]; break;$

9 **endif**

10 $Tmp \leftarrow \cup_{y' \in \chi} V(i-1,y').$GET-LAST$();$

11 **for** *each* $te = \{score, label, prev\} \in Tmp$ **do**

12 $te.score \leftarrow te.score +$ MR$(x_i, te.label, y');$

13 **endfor**

14 *sort Tmp by score desc*; $max \leftarrow Tmp[1];$

15 **for** $j = i-2; j >= 0; j--$ **do**

16 **if** $max.prev.label =$ 'don't care' **then**

17 $max \leftarrow U(j, max.label)[max.prev.idx];$

18 $label[j] \leftarrow \chi;$

19 **else**

20 $max \leftarrow V(j, max.prev.label)[max.prev.idx];$

21 $label[j] \leftarrow max.prev.label;$

22 **endif endfor**

23 $score \leftarrow$ MARGINALIZE$(i, labels);$

24 $UT(i,y) \leftarrow UT(i,y) \cup \{score, y, Tmp[1].prev\};$

25 GET-NEXT-V$(V, i-1, Tmp[1].label, \chi);$

26 **endfor**

Figure A.3. Algorithm for computing the next highest-probability entry in $U(i,y)$ used in TOP1PROBPROJECT().