Interactive Data Management and Data Analysis

by

Ying Yang

May, 2017

A dissertation submitted to the

Faculty of the Graduate School of the

State University of New York at Buffalo

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science and Engineering

Interactive Data Management and Data Analysis

by

Ying Yang

Submitted to the Department of Computer Science and Engineering
on May, 2017, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Everyone today has a big data problem. Data is everywhere and in different formats, they can be referred to as data lakes, data streams, or data swamps. To extract knowledge or insights from the data or to support decision-making, we need to go through a process of collecting, cleaning, managing and analyzing the data. In this process, data cleaning and data analysis are two of the most important and time-consuming components.

One common challenge in these two components is a lack of interaction. The data cleaning and data analysis are typically done as a batch process, operating on the whole dataset without any feedback. This leads to long, frustrating delays during which users have no idea if the process is effective. Lacking interaction, human expert effort is needed to make decisions on which algorithms or parameters to use in the systems for these two components.

We should teach computers to talk to humans, not the other way around. This dissertation focuses on building systems — Mimir and CIA — that help user conduct data cleaning and analysis through interaction. Mimir is a system that allows users to clean big data in a cost- and time-efficient way through interaction, a process I call on-demand ETL. Convergent inference algorithms (CIA) are a family of inference algorithms in probabilistic graphical models (PGM) that enjoys the benefit of both exact and approximate inference algorithms through interaction.

Mimir provides a general language for user to express different data cleaning needs. It acts as a shim layer that wraps around the database making it possible for the bulk of the ETL process to remain within a classical deterministic system. Mimir also helps users to measure the quality of an analysis result and provides rankings for cleaning tasks to improve the result quality in a cost efficient manner. CIA focuses on providing user interaction through the process of inference in PGMs. The goal of CIA is to free users from the upfront commitment to either approximate or exact inference, and provide user more control over time/accuracy trade-offs to direct decision-making and computation instance allocations. This dissertation describes the Mimir and CIA frameworks to demonstrate that it is feasible to build efficient interactive data management and data analysis systems.

Acknowledgments

First of all, I would like to thank my advisor Dr. Oliver Kennedy, for supporting me over the years and giving me excellent guidance in the area of database. Dr. Kennedy inspires me lots of thinking and learning all over the years. He gives me lots of freedom to explore new things and ideas. I would like to thank Dr. Jan Chomicki for supporting me and leading me into research at the beginning of my Ph.D. study. I would like to thank Dr. Bharat Jayaraman for serving as my committee member and asking many thoughtful questions.

Mimir would have not been possible without all its users. I would like to thank all the scientists and students who have interacted with me, without whose generous sharing of knowledge we could not have built Mimir to its current state. A very incomplete list includes Dieter Gawlick, Zhen Hua Liu, Ronny Fehling, Boris Glavic, Beda Hammerschmidt, Eric S. Chan, Adel Ghoneimy, Ying Lu, Arindam Nandi, Niccolo Meneghetti, William Spoth, Poonam Kumari, Lisa Lu, Mike Brachman, Aaron Huber.

I would also like to thank my friends and colleagues for the help, insightful discussion, and happy moments spending together. I would like to thank Ladan Golshan Ara, Niccolo Meneghetti, Yingbo Zhou, Danyang Chen, Weida Zhong, Ning Deng.

Last, I would like to thank my parents Kai Yang, Wei Wang for all their love and support during my Ph.D. study. I would like to thank my boyfriend, Zhen Xu, for the accompany and support all over the years.

My graduate study has been supported by NPS Award #N00244-16-1-0022, NSF Award #1409551 and #1640864, and by gifts from Oracle Academic Relations. Opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, the Naval Postgraduate School, or Oracle.

Software, Data and Code

- Mimir is available at `http://mimirdb.info`. It is a team effort to further develop and maintain this system.
- The code for the prototype systems is avaiable at `https://github.com/UBOdin/mimir`.
- The code for the prototype systems we developed to study each component of Mimir is also available separately.
    1. Lens Composition (Chapter 3.5.4):`http://www.cse.buffalo.edu/~yyang25/code.html`
    2. Pay-as-you-go Data Cleaning (Chapter 3.5.5):`http://www.cse.buffalo.edu/~yyang25/code.html`
    3. Flexibility, accuracy and convergence time for Convergent Inference Algorithms (Chapter 5.4):`http://www.cse.buffalo.edu/~yyang25/code.html`
- Most data that we can legally release are available at: `http://www.cse.buffalo.edu/~yyang25/data.html`
- This introductory vidoe in our YouTube channel is related to this dissertation: `https://www.youtube.com/watch?v=jow4JmDOxPs`
- This poster is related to this dissertation: `http://www.cse.buffalo.edu/~yyang25/p2322-yang_poster.pdf`

Contents

List of Figures

Chapter 1

Introduction

Nowadays data is everywhere. Organizations or individuals accumulate humongous data that they want to access and analyze as a consolidated whole. There are several components in this process. First, **collecting data sets:** data is extracted from multiple sources: Internet of things (IoT), databases and so on. Second, **data cleaning:** the collected data often has inconsistencies in schema, formats and adherence to constraints, due to many factors including data entry errors and merging from multiple sources. Third: **data management systems:** the cleansed data is usually represented in a new or transformed way as tables or files in a central data warehouse or in a data lake. Finally, **data analysis:** an analyst queries the data to do any actual analysis. Different algorithms can be applied for different purposes: classification, clustering, inference in graphical models, knowledge construction and so on. Each component plays an important role and my thesis aims at providing time- and cost-efficient frameworks for the components by providing interactions with user. In this thesis, I focus on **data cleaning** and **data analysis**, the reasons are explained and motivated throughout this chapter.

Effective analytics depends on analysts having access to accurate, reliable, high-

quality information. Data cleaning is a time-consuming task. A new survey [2] of data scientists found that they spend most of their time cleaning rather than mining or modeling data. According to the survey, around 80% of their time is spent on data preparation, including data cleaning and collecting data sets. Data cleaning has received tremendous interest from both academia [33,37,91,101,102] and industry [36, 53,126]. However, many of existing work have two problems: (1) **Focusing only on cleaning accuracy:** a lot of existing works are focusing on improving the precision of data cleaning task ignoring the fact that the use of complicated algorithms to improve accuracy causes data cleaning a time-consuming task. (2) **Lack of interactivity:** the cleaning process is typically done as a batch process, operating on the whole dataset without any feedback. This leads to long, frustrating delays during which users have no idea if the cleaning is effective.

An on-demand data cleaning solution [20, 31, 71, 73] has been proposed, cleaning data incrementally by interacting with the user for feedback. However, these works usually focus on only one or two specific problems in data cleaning. For example, Jeffery et al. [71] provides an on-demand data cleaning solution for data with schema matching and entity resolution problems. Chai et al. [31] focuses on information extraction and entity resolution. Each solution uses a specific data structure to represent the data and uses different factors to rank cleaning tasks. In real world, raw data can contain multiple problems. In this case, manually "gluing" these works can cause lot of extra work for user.

In this thesis, I propose a general on-demand data cleaning framework - **Mimir**. One key requirement for such a data cleaning system is to be able to solve all different problems in data cleaning process in a time- and cost-efficient way through interaction with user. The challenges introduced by this requirement are: (1) How to provide a language for user to express different data cleaning needs. (2) How to measure the

quality of the data in terms of uncertainty. (3) How to rank cleaning task to improve data quality with minimal cost. (4) How to interact with user efficiently. Details on how Mimir solves these challenges are discussed in Chapter 2 and Chapter 3.

The second component this thesis focuses on is data analysis. There are a lot of algorithms used in data analysis, and inference in probabilistic graphical model (PGM) is one of them. PGMs are a factorized encoding of joint (multivariate) probability distributions. Even large distributions can often be compactly represented as a PGM. Inference is a common operation which reconstructing the marginal probability for a subset of the variables in the full joint distribution.

The reasons that I choose to focus on inference on PGMs are: **Popularity:** (1) PGMs involve the uncertainty in the modeling that cause it to less sensitivity to noise phenomenon. (2) The graph structure of PGM provides interpretability of the result. **Many applications in many domains:** there are a lot of applications based on inference on PGMs: expert system, natural language processing, handwriting, video recognition, bio-sequence analysis, knowledge construction and so on. **Closely related to databases:** over the past decade, a class of model database systems have begun to add support for PGMs within database engines, allowing graphical models to be queried through SQL [40, 108, 123], combined with other data for joint analysis [69, 76], or used for analytics over messy data [91, 118, 120].

Existing inference algorithms are either exact or approximate. Exact algorithms [29] like variable elimination and belief propagation produce exact results, but can be slow. On the other hand, approximate algorithms [125, 127] like Gibbs sampling generate estimates within any fixed time bounds, but only converge asymptotically to exact results.

There are several problems in current inference solutions: (1) **Lack of accurate interactivity:** similar to data cleaning, data analysis is a time-consuming task too.

The complexity of inference grows exponentially with increasing dimensionality of the data. Exact inference will not reveal any result until obtaining the exact solution. This also leads to long, frustrating delays during which users have no idea when algorithm will terminate. Approximate inference can provide a sort of interaction by providing current approximate result with a confidence bound. However, these bounds are theoretical and not accurate enough to direct decision-makings or computation instance allocations.

(2) **Need for much user effort:** the model database systems typically employ approximate inference techniques, as model complexity can vary widely with different usage patterns and responsiveness is typically more important than exact results. However, exact inference can sometimes produce an exact result *faster* than it takes an approximate algorithm to converge, even for moderately complex inference problems. Furthermore, in interactive settings, the user may be willing to wait for more accurate results. In either case, the choice of whether or not use an exact algorithm must wait until the system has already obtained an approximation or be judged by user, which is a difficult decision especially for non-experts.

In this thesis, I explore a family of convergent inference algorithms (CIAs) that simultaneously acts as both approximate and exact inference algorithms: Given a fixed time bound, a CIA can produce a bounded approximate inference result, but will also terminate early if it is possible to converge to an exact result. Like a file copy progress bar, CIAs can provide a "result accuracy progress bar" that is guaranteed to complete eventually. Similar to online-aggregation (OLA) [63], CIAs give users and client applications more control over accuracy/time trade-offs and do not require an upfront commitment to either approximate or exact inference.

The challenges introduced by such inference algorithms are: (1) How to guarantee convergence in a limited time. (2) How to guarantee to provide a confidence bound

during the inference process. (3) How to guarantee that CIAs have time complexity that is competitive with classic exact inference algorithms. (4) How to guarantee that the approximation accuracy is competitive with common approximate techniques. Details on how CIAs solve these challenges are discussed in Chapter 4 and Chapter 5.

In this chapter, I first present examples of data cleaning scenarios and then, I present the goal of my study, and the technical contributions.

1.1 Sample Target Senario for Data Cleaning

I present examples of data cleaning systems to illustrate the target workload I focus on in this dissertation. The goal of this section is not to provide a complete description of the workload but instead to provide enough information to guide the reader through the rest of the thesis. A detailed description and examples of data cleaning workload are the topics of Chapter 3.

Example 1 *Alice is an analyst at the HappyBuy retail store, and is developing a promotional strategy based on public opinion ratings for its products gathered by two data collection companies. A thorough analysis of the data requires substantial data-cleaning effort from Alice: As shown in Figure 1-1, the rating companies' schemas are incompatible, and HappyBuy's own product data is incomplete. However, Alice's preliminary analysis is purely exploratory, and she is hesitant to invest the full effort required to curate this data.*

The upfront costs of cleaning have lead many to instead in-line cleaning tasks into the analytical process, so that only immediately relevant cleaning tasks are performed. This solution uses data lake or NoSQL databases. A data lake is a storage repository that holds a vast amount of raw data in its native format until it is needed. While

a hierarchical data warehouse stores data in files or folders, a data lake uses a flat architecture to store data. In this way, users don't need to spend a lot of time building the central data warehouse and a data lake provides more flexibility as the data may be needed in various formats.

Example 2 *Alice realizes that she only needs two specific attributes for her analysis:* `category` *and* `rating`*. Therefore from the data lake, she considers manually constructing a task-specific data set containing a sanitized version of only these two columns.*

This deferred approach is more lightweight, but encourages analysts to develop brittle, one-off data cleansing solutions, incurring significant duplication of effort or organizational overheads. A third approach, initially explored as part of Paygo [71], instead cleans data incrementally in response to specific query requirements. This form of *on-demand cleaning* results in a sanitized data set that is based on a principled trade-off between the quality desired from the data set and the human effort invested in cleaning it. Paygo specifically targets two cleaning tasks: schema matching and entity resolution, and other systems have since appeared for schema matching [11], as well as other tasks like information extraction [32], and inference [123, 128].

Example 3 *Alice decides to obtain an accurate result for products with good rating information. So she needs to solve two cleaning problems: (1) Schema matching: she needs to merge rating information from the two companies together, but they use different schemas to present the data. (2) Missing values: she still needs to fix values in* `category` *and* `rating`*.*

She first uses Paygo to fix the schema matching problem. Paygo uses a triple (object, attribute, value) to represent one entity, therefore data value and schema are consider in a uniform fashion. For example, for an entity in Table "Ratings

Product				
id	**name**	**brand**	**category**	**ROWID**
P123	Apple 6s, White	?	phone	R1
P124	Apple 5s, Black	?	phone	R2
P125	Samsung Note2	Samsung	phone	R3
P2345	Sony to inches	?	?	R4
P34234	Dell, Intel 4 core	Dell	laptop	R5
P34235	HP, AMD 2 core	HP	laptop	R6

Ratings1				
pid	...	**rating**	**review_ct**	**ROWID**
P123	...	4.5	50	R7
P2345	...	?	245	R8
P124	...	4	100	R9

Ratings2				
pid	...	**evaluation**	**num_ratings**	**ROWID**
P125	...	3	121	R10
P34234	...	5	5	R11
P34235	...	4.5	4	R12

Figure 1-1: Incomplete example relations, annotated with implicit per-row lineage markers (ROWID).

1", $< P123, rating, 4.5 >$, attribute "rating" is a schema name and object and value "P123" and "4.5" are data values. Then she uses the BayesStore system [123] to solve missing values in Product table. BayesStore stores data in a normal relational table in database. Also, Paygo has a ranking mechanism for cleaning tasks which BayesStore doesn't.

From the example, we can see that without a uniform framework, on-demand data cleanings can be challenging. A typical data cleaning process often involves many distinct cleaning tasks, requiring that multiple on-demand data cleaning systems be used in tandem. However, the data representations and quality metrics used by these systems are optimized for very specific use-cases, making composition difficult.

1.2 Technical Contributions

I worked on two systems that focus on providing interactive solutions for data cleaning and data analysis.

In Mimir, I explored and addressed the challenges of composing specialized on-demand data cleaning techniques into a general-purpose workflow. The result is **a unified model for on-demand cleaning called On-Demand ETL (Exact, transform and load)** that bridges the gap between these systems and allows them to be gracefully incorporated into existing ETL and analytics workflows. This unified model builds around ordinary SQL, retaining compatibility with existing standards for ETL design, data analysis, and database management.

The contributions include:

Representing Incomplete Data. On-demand cleaning permits trade-offs between data quality, and the effort needed to obtain high-quality data. This requires a representation for the quality loss incurred by only partially cleaning data. Existing on-demand cleaning systems use specialized, task-specific representations. In Section 2.4 I describe an existing representation for incomplete information called PC-Tables [55, 56, 68], and show how it can be leveraged by On-Demand ETL.

Expressing Composition. If the output of a cleaning technique is non-deterministic, then for closure, it must accept non-deterministic input as well. In Section 3.1, I define a model for non-deterministic operators called lenses that capture the semantics of on-demand data cleaning processes. I illustrate the generality of this model through examples, and show that it is closed over PC-Tables.

Backwards Compatibility. For On-Demand ETL to be practical, it must be compatible with traditional data management systems and ETL pipelines. In Section 3.2, I develop a practical implementation of PC-Tables [68] called Virtual C-Tables that

can be safely embedded into a classical, deterministic database system or ETL workflow.

Presenting Data Quality. In Section 3.3, I discuss how to present the quality loss incurred by incomplete cleaning to end-users. I show how lightweight summaries can be used to alert an analyst to specific problems that affect their analysis, and how On-Demand ETL computes a variety of quality measures for query results.

Feedback. Section 3.4 highlights how Virtual C-Tables act as a form of provenance, linking uncertainty in query outputs to the lenses that created them. These links allow for lens-defined cleaning tasks that improve the quality of query results. I introduce a concept called *the cost of perfect information* (CPI) that relates the value of a cleaning task that improves a result's quality, to the cost of performing the task, allowing cleaning tasks to be ranked according to their net value to the analyst.

Experimental Results. Finally, in Section 5.4, I present experimental results that demonstrate the feasibility of On-Demand ETL and provide several insights about its use.

In CIA, I explored a family of *convergent inference algorithms* (CIAs) that simultaneously act as both approximate and exact inference algorithms: Given a fixed time bound, a CIA can produce a bounded approximate inference result, but will also terminate early if it is possible to converge to an exact result.

Detailed contribution includes:

Concept of Convergent Inference Algorithms. I propose a new family of Convergent Inference Algorithms (CIAs) that provide approximate results over the course of inference, but eventually converge to an exact inference result.

A Naive Convergent Inference Algorithm. I cast the problem of Convergent Inference as a specialization of Online Aggregation, and propose a naive, *constant-*

space convergent inference algorithm based on Linear Congruential Generators. This naive CIA guarantees to provide a confidence bound during the inference process and grarantee to converge to an exact result within limited time.

Leaky Joins. To satisfy the challenge of competitive time complexity and approximate accuracy with existing solutions, I proposed Leaky Joins, a novel Online Aggregation algorithm specifically designed for Convergent Inference with competitive time complexity and approximate accuracy.

Proof of Complexity. I show that Leaky Joins have time complexity that is no more than one polynomial order worse than classic exact inference algorithms, and provide an $\epsilon-\delta$ bound to demonstrate that the approximation accuracy is competitive with common approximation techniques.

Experiment Results. I present experimental results on both synthetic and real-world graph data to demonstrate that (a) Leaky Joins gracefully degrade from exact inference to approximate inference as graph complexity rises. (b) Leaky Joins have exact inference costs competitive with classic exact inference algorithms, and approximation performance competitive with common sampling techniques.

Lessons learned. I discuss lessons learned in my attempts to design a convergent inference algorithm using state-of-the-art incremental view maintenance systems [10, 81].

Chapter 2

Preliminaries for Mimir

2.1 ETL

Extract, Transform, Load (ETL) in Figure 2-1 is a process in data warehousing responsible for pulling data out of the source systems and placing it into a data warehouse or data lake. While a hierarchical data warehouse stores data in files or folders, a data lake uses a flat architecture to store data. ETL involves the following tasks:

1. **Data extraction** is where data is extracted from homogeneous or heterogeneous data sources, data with identical or different structures.

2. **Data transformation** is where the data is transformed for storing in the desired format or structure for the purposes of querying and analysis. This process may involve the following tasks:

    - *Applying business rules.* For example, calculating new measures and dimension,

    - *Cleaning.* For example, mapping NULL to 0 or "Male" to "M" and "Female" to "F",

- *Filtering.* For example, selecting only certain columns or rows to load,
- *Splitting a column into multiple columns or merging multiple columns into one column*,
- *Joining together data from multiple sources.* For example, schema matching or shema mapping,
- *Transposing rows and columns*, such as transforming row wise data into column wise,
- *Applying any kind of data validation.* For example, if the first 3 columns in a row are empty, then reject the row from processing.

3. **Data loading** is where the data is loaded into the final target database, more specifically, an operational data store, data mart, or data warehouse.

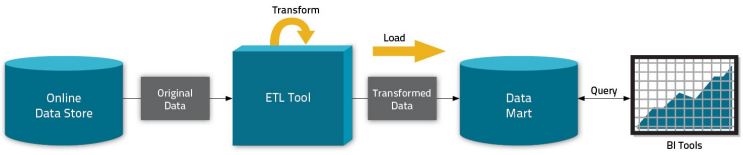


Figure 2-1: Traditional ETL and Business Intelligence Process

ETL processing is a very time-consuming task. Data warehouses are typically assembled from a variety of data sources with different formats and purposes. As such, ETL is a key process to bring all the data together in a standard, homogeneous environment. A lot of existing work focuses on obtaining accurate data through ETL processing. However, since some ETL systems have to scale to process terabytes of data to update data warehouses with tens of terabytes of data. Using complicated algorithms for ETL processing increases the processing time tremendously. There is a trade-off between time and accuracy. Mimir is one of the systems that provides user more control over this trade-off.

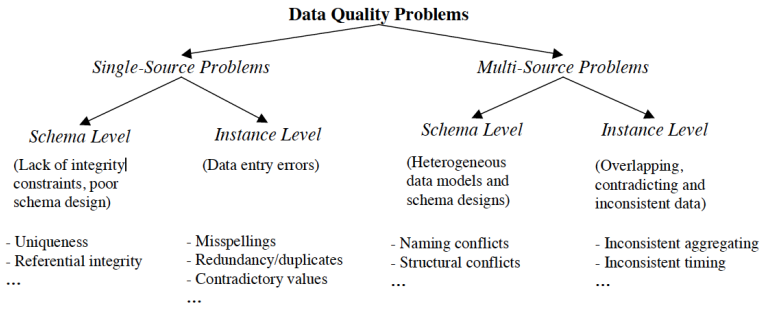


Figure 2-2: Classification of data quality problems [101]

2.2 Data Cleanig

Data cleaning, also called data curation, wrangling or scrubbing, deals with detecting and removing errors and inconsistencies from data in order to improve the quality of data. Data cleaning [101] is especially required when integrating heterogeneous data sources. Heterogeneous data sources include Data model heterogeneous (Different ways of representing and storing the same data) and Semantic heterogeneous (Data across constituent databases may be related but different). In data warehouses, data cleaning is a major part of the ETL process. Since data warehouses load and continuously refresh huge amounts of data from a variety of sources, so the probability that some of the sources contain "dirty data" is high. Since data warehouses are used for decision making, the correctness of their data is vital to avoid wrong conclusions. For instance, duplicated or missing information will produce incorrect or misleading statistics ("garbage in, garbage out").

In this section, I follow the classification convention in Erhard et al. [101] and discuss briefly major data quality problems to be solved by data cleaning processes. These classifications are shown in Figure 2-2.

2.2.1 Single-source problems

Unlike data sources without schema such as files, database systems enforce restrictions and specific data models. For example, relation databases can require integrity constraints as well as application-specific integrity constraints. Table 2-3 shows some examples of single-source problems at schema level, they are mainly due to the violation of integrity constraints of relational database.

Scope	Problem	Dirty Data	Reasons/Remarks
Attribute	Illegal values	bdate=30.13.70	values outside of domain range
Record	Violated attribute dependencies	age=22, bdate=12.02.70	age = (current date - birth date) should hold
Record type	Uniqueness violation	emp1=(name="John Smith", SSN="123456") emp2=(name="Peter Miller", SSN="123456")	uniqueness for SSN violated
Source	Referential integrity violation	emp=(name="John Smith", deptno=127)	referenced department (127) not defined

Figure 2-3: Examples for single-source problems at schema level (violated integrity constraints)

For both schema- and instance-level problems, we can classify different problem scopes: attribute (field), record, record type and source. Examples for the various cases are shown in Tables 2-3 and 2-4. Note that uniqueness constraints specified at the schema level do not prevent duplicated instances. For example, information on the same real world entity may be entered twice with different attribute values (see example in Table 2-4).

Scope	Problem	Dirty Data	Reasons/Remarks
Attribute	Missing values	phone=9999-999999	unavailable values during data entry (dummy values or null)
Record	Violated attribute dependencies	city="Redmond", zip=77777	city and zip code should correspond
Record type	Duplicated records	emp1=(name="John Smith",...); emp2=(name="J. Smith",...)	same employee represented twice due to some data entry errors
Source	Wrong references	emp=(name="John Smith", deptno=17)	referenced department (17) is defined but wrong

Figure 2-4: Examples for single-source problems at instance level

2.2.2 Multi-source problems

The problems present in single sources are aggravated when multiple sources need to be integrated. Since the sources are typically developed, deployed and maintained independently to serve specific needs, each source may contain dirty data and the data in the sources may be represented differently, overlap or contradict.

Customer

CID	Name	Street	City	Sex
11	Kristen Smith	2 Hurley Pl	South Fork, MN 48503	0
24	Christian Smith	Hurley St 2	S Fork MN	1

Client

Cno	LastName	FirstName	Gender	Address	Phone/Fax
24	Smith	Christoph	M	23 Harley St, Chicago IL, 60633-2394	333-222-6542 /333
493	Smith Kris L.	F	2 Hurley Place, South Fork MN, 48503-5998	444-555-6666	

Customers (integrated target with cleaned data)

No	LName	FName	Gender	Street	City	State	ZIP	Phone	Fax	CID	Cno
1	Smith	Kristen L.	F	2 Hurley Place	South Fork	MN	48503	444-555-6666		11	493
2	Smith	Christian	M	2 Hurley Place	South Fork	MN	48503			24	
3	Smith	Christoph	M	23 Harley Street	Chicago	IL	60633	333-222-6542	333		24

Figure 2-5: Examples of multi-source problems at schema and instance level

At the schema level, in order to integrate different sources, data model and schema design differences need to be addressed by a process called schema matching. The main problems in schema matching are naming and strutural conflicts. Naming conflicts arise when the same name is used for different objects (homonyms) or different names are used for the same object (synonyms). Structural conflicts occur in many variations and refer to different representations of the same object in different sources, like attribute vs. table representations, different component structure, different data types, different integrity constraints, and so on.

In addition to schema-level conflicts, many conflicts appear only at the instance level (data conflicts). All the same problems from the single-source case can also occur with different representations in different sources (e.g., duplicated records, contradicting records). A main problem for cleaning data from multiple sources is to identify overlapping data, and in particular matching records referring to the same real-world entity (e.g., customer). This problem is also referred to as duplicate elimination or entity resolution.

The two sources in the example of Figure 2-5 are both in relational format but exhibit schema and data conflicts. At the schema level, there are name conflicts like Customer/Client, Cid/Cno, Sex/Gender, and structural conflicts like different representations for names and addresses. At the instance level, there are different gender representations (“0”/“1” vs. “F”/“M”) and presumably a duplicate record (Kristen

Smith). The latter observation also reveals that while Cid and Cno are both source-specific identifiers, their contents are not comparable between the sources. Different numbers (11/493) may refer to the same person while different persons can have the same number (24). The third table shows a possible solution.

2.3 Data Cleaning Approaches

Data Cleaning is an important topic in database area and has been extensively studied. In this section, I refer to some existing solutions to problems in data cleaning process.

2.3.1 Information Extraction

Information extraction (IE) is the task of automatically extracting structured information from unstructured and/or semi-structured documents. Probabilistic databases are often used to work for information extraction [60, 119, 121, 122]. Chai et al. [31] propose a solution to directly provide feedback and information extraction and integration to automatically process such feedback.

2.3.2 Schema Matching

A survey and its follow-up [23, 100] present a comprehensive classification of schema matching approaches. A rewrite-based optimization approach for schema matching process is introduced in [99]. Doan et al. [43] propose a machine-learning approach for schema matching. Chapter 9 in [22, 58] provides a generic overview of the existing efforts on bechmarking schema matching and mapping tasks. Schema matching systems has been developed [42, 87, 89]. Incremental schema matching methods are

introduced in [22, 24, 47, 71]. Zhang et al. [129] explore the use of crowdsourcing to reduce the uncertainty of schema matching.

2.3.3 Entity Resolution

Elmagarmid et al. [48] is a survey for duplicate detection. It focuses on the problem of lexical heterogeneity and surveys techniques for addressing this problem. Chaudhuri et al. [33] presents tuple matching in an online data cleaning senario. Singla et al. [112] propose an integrated solution to the entity resolution problem based on Markov logic. Hanna et al. [83] comparatively evaluate several entity resolution approaches. Incremental entity resolution [13, 57] allow users to explore a trade-off between the resolution cost and the achieved quality of the resolved data. Panse et al. [97] present an indeterministic approach for deduplication by using a probabilistic target model. Altwaijry et al. [14] explore “on-the-fly" data cleaning in the context of query.

2.3.4 Domain Constraint Repair

Potter’s Wheel [102] is an interactive data cleaning system that tightly integrates transformation, infers patterns and detects discrepancy from the pattern. MauveDB [40] defines model-based view acting as an “independence" layer between raw data and userapplication and effieciently maintains models when raw data is updated. ER-ACER [91] is an iterative statistical framework for inferring missing information and correcting such errors automatically. Finally, BayesStore [123] employs concise statistical relational models to effectively encode the correlation patterns between uncertain data, and promotes probabilistic inference and statistical model manipulation as part of the standard DBMS operator repertoire to support effecient query processing.

2.3.5 Archival

Kennedy et.al. [78] characterize contextual dependence errors from ETL process and business intelligence and explore several strategies for efficiently detecting and quantifying the effects of contextual dependence on query outputs.

2.4 Probabilistic Query Processing

A deterministic database is a finite collection of relation instances $\{R_1, \ldots, R_k\}$ over a schema $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_k\}$. According to the "possible worlds" semantics [114] a probabilistic database $\mathcal{D}$ consists of a pair $(\mathbf{W}, P)$, where $\mathbf{W}$ is a large collection of deterministic databases, the so called possible worlds, all sharing the same schema $\mathcal{S}$, and P is a probability measure over $\mathbf{W}$. Roughly speaking, $\mathcal{D}$ is a database whose schema is known but whose internal state is uncertain, and $\mathbf{W}$ simply enumerates all its plausible states. Denote by R the set of all tuples that appear in some possible world (often called possible tuples). Each element of R is an *outcome* for the probability space $(\mathbf{W}, P)$. The *confidence* of a possible tuple t is simply the probability that it will appear in the database $\mathcal{D}$, i.e. its marginal probability

$$P(t \in \mathcal{D}) = \sum_{W_i \in \mathbf{W} | t \in W_i} P(W_i)$$

The goal of probabilistic databases [9,28,54,67,70,75,95,111] is to support the execution of deterministic queries like regular, deterministic databases do. Denote by Q an arbitrary deterministic query (i.e., a query expressible in classical bag-relational algebra) and by $sch(Q)$ the schema defined by it, which consists of a single relation. The application of Q to $\mathcal{D}$, denoted by $Q(\mathcal{D})$, generates a new probability space $(\mathbf{W}', P')$

$$
\begin{aligned}
e &:= \mathbb{R} \mid \mathit{Column} \mid \texttt{if } \phi \texttt{ then } e \texttt{ else } e \\
&\quad \mid e \ \{+, -, \times, \div\} \ e \mid \mathit{Var}(\mathit{id}[, e[, e[, \ldots]]]) \\
\phi &:= e \ \{=, \neq, <, \leq, >, \geq\} \ e \mid \phi \ \{\wedge, \vee\} \ \phi \mid \top \mid \bot \\
&\quad \mid e \texttt{ is null} \mid \neg\phi
\end{aligned}
$$

Figure 2-6: Grammars for boolean expressions ϕ and numerical expressions e including support for VG-Functions $\mathit{Var}(\ldots)$.

where $\mathbf{W}' = \{Q(W_i) \mid W_i \in \mathbf{W}\}$ and

$$P'(t \in Q(\mathcal{D})) = \sum_{W_i \in \mathbf{W} | t \in Q(W_i)} P(W_i)$$

A probabilistic query processing (PQP) system is supposed to answer a deterministic query Q by listing all its possible answers and annotating each tuple with its marginal probability, or by computing expectations for aggregate values. These tasks are difficult in practice, mainly for two reasons: (i) $\mathbf{W}$ is usually too large to be enumerated explicitly, and (ii) computing marginals is provably #P-hard in the general case. For example, if a schema contains a single relation and the set of possible worlds contains all subsets of a given set of 100 tuples, then there are 2^{100} distinct possible worlds where each possible tuple appears in half.

One way to make probabilistic query processing efficient is to encode $\mathbf{W}$ and P with a compact, factorized representation. In this thesis I adopt a generalized form of C-Tables [68, 75] to represent $\mathbf{W}$, and PC-Tables [55, 56] to represent the pair $(\mathbf{W}, P)$. A C-Table [68] is a relation instance where each tuple is annotated with a lineage formula ϕ, a propositional formula over an alphabet of variable symbols Σ. The formula ϕ is often called a *local condition* and the symbols in Σ are referred to as *labeled nulls*, or just variables. Intuitively, for each assignment to the variables in Σ we obtain a possible relation containing all the tuples whose formula ϕ is satisfied.

For example:

Product					
	pid	**name**	**brand**	**category**	ϕ
t_1	P123	Apple 6s, White	Apple	phone	$x_1 = 1$
t_2	P123	Apple 6s, White	Cupertino	phone	$x_1 = 2$
t_3	P125	Samsung Note2	Samsung	phone	$\top$

The above C-Table defines a set of three possible worlds, $\{t_1, t_3\}$, $\{t_2, t_3\}$, and $\{t_3\}$, i.e. one world for each possible assignment to the variables in the one-symbol alphabet $\Sigma = \{x_1\}$. Notice that no possible world can have both t_1 and t_2 at the same time. C-Tables are closed w.r.t. positive relational algebra [68] : if $\mathbf{W}$ is representable by a C-Table and Q is a positive query then $\mathbf{W}' = \{Q(W_i) \mid W_i \in \mathbf{W}\}$ is representable by another C-Table.

Following the approach of PIP [75], in this thesis I adopt VG-RA (variable-generating relational algebra), a generalization of positive bag-relation algebra with extended projection, that uses a simplified form of VG-functions [70]. In VG-RA, VG-functions (i) dynamically introduce new Skolem symbols in Σ, that are guaranteed to be unique and deterministically derived by the function's parameters, and (ii) associate the new symbols with probability distributions. Hence, VG-RA can be used to define new C-Tables. Primitive-valued expressions in VG-RA (i.e., projection expressions and selection predicates) use the grammar summarized in Figure 2-6. The primary addition of this grammar is the VG-Function term: $Var(\ldots)$.

From PIP, I also inherit a slightly generalized form of C-Tables. Our C-Tables differ from the canonical ones in the following: (i) Variables in Σ are allowed to range over continuous domains, (ii) attribute-level uncertainty is encoded by replacing missing values with VG-RA *expressions* (not just functions) that act as Skolem terms and (iii) these VG-RA expressions allow basic arithmetic operations. The previous example is equivalent to the PIP-style C-Table:

Product			
pid	**name**	**brand**	**category**
P123	Apple 6s, White	$Var('X', \texttt{R1})$	phone
P125	Samsung Note2	Samsung	phone

Where R_1 is the ROWID of row with pid = p123. From now on, without explicitly mentioning PIP, I will assume all C-Tables support the generalizations discussed above. It has been shown that C-Tables are closed w.r.t VG-RA [68, 75]. The semantics for VG-RA query evaluation $[[\cdot]]_{CT}$ over C-Tables [67, 68, 75] are summarized in Figure 2-7. These semantic rules make extensive use of the *lazy evaluation* operator $[[\cdot]]_{lazy}$, which uses a *partial* binding of $Column$ or $Var(\ldots)$ atoms to corresponding expressions. Lazy evaluation applies the partial binding and then reduces every sub-tree in the expression that can be deterministically evaluated. Non deterministic sub-trees are left intact. Any tuple attribute appearing in a C-Table can be encoded as an abstract syntax tree for a partially evaluated expression that assigns it a value. This is the basis for evaluating projection operators, where every expression e_i in the projection's target list is lazily evaluated. Column bindings are given by each tuple in the source relation. The local condition ϕ is preserved intact through the projection. Selection is evaluated by combining the selection predicate ψ with each tuple's existing local condition ϕ. As an optimization, tuples for which ϕ deterministically evaluates to false ($\bot$) are preemptively discarded.

Expression	**Evaluates To**
$[[\pi_{a_i \leftarrow e_i}(R)]]_{CT}$	$\{ \langle a_i : [[e_i(t)]]_{lazy}, \phi : t.\phi \rangle \mid t \in [[R]]_{CT} \}$
$[[\sigma_{\psi}(R)]]_{CT}$	$\{ \langle a_i : t.a_i, \phi : [[t.\phi \wedge \psi(t)]]_{lazy} \rangle \mid t \in [[R]]_{CT} \wedge ([[t.\phi \wedge \psi(t)]]_{lazy} \not\equiv \bot) \}$
$[[R \times S]]_{CT}$	$\{ \langle a_i : t_1.a_i, a_j : t_2.a_j, \phi : t_1.\phi \wedge t_2.\phi \rangle \mid t_1 \in [[R]]_{CT} \wedge t_2 \in [[S]]_{CT} \}$
$[[R \uplus S]]_{CT}$	$\{ \langle a_i : t.a_i, \phi : t.\phi \rangle \mid t \in ([[R]]_{CT} \uplus [[S]]_{CT}) \}$

Figure 2-7: Evaluation semantics for positive bag-relational algebra over C-Tables

A PC-Table [55, 56] is a C-Table augmented with a probability measure P over the possible assignments to the variables in Σ. Since each assignment to the variables in Σ generates a possible world, a PC-Table induces a probability measure over $\mathbf{W}$.

Hence, it can be used to encode a probabilistic database $(\mathbf{W}, P)$. PC-Tables are the foundation for several PQP systems, including MayBMS [67], Orchestra [54] and PIP [75]. Green et al. [55] observed that PC-Tables generalize other models like Trio [9]. The relationship between PC-Tables and VG-RA is discussed in further detail in Section 3.1.

2.5 On-Demand Data Cleaning Tools

There are numerous tools for on-demand data curation, each targeting *specific* challenges like schema matching [11, 71], de-duplication [71], information extraction [30, 32], information integration [21], or ontology construction [15, 59]. On-Demand ETL generalizes these, providing a tool for on-demand data curation that these solutions can be plugged into. On-demand curation can also be thought of as a highly-targeted form of crowd-sourced databases [90], which leverage the power of humans to perform complex tasks.

The problem of incomplete data arises frequently in distributed systems, where node failures are common. Existing solutions based on uncertainty [27, 52, 85, 117] can similarly be expressed in On-Demand ETL to provide more fine-grained analyses of result quality over partial data than these approaches provide natively.

2.6 Prioritizing Feedback

Prioritizing curation tasks, as addressed in Section 3.4, is quite closely related to Stochastic Boolean Function Evaluation, or SBFE, where the goal is to determine the value of a given Boolean formula by paying a price to discover the exact value of uncertain boolean variables. The problem is hard in its general form; exact so-

lutions and heuristics have been proposed for several classes of functions [74, 116]. More recently Deshpande et al. [39] designed a 3-approximation algorithm for linear threshold formulas, while Allen et al. [12] developed exact and approximate solutions for monotone k-term DNF formulas.

Chapter 3

Interactive Data Cleaning for Data Management - Mimir

3.1 Lenses

A lens is a data processing component that is evaluated as part of a normal ETL pipeline. Unlike a typical ETL processing stage that produces a single, deterministic output, a lens instead produces a PC-Table $(\mathbf{W}, P)$, which defines the set of possible outputs, and a probability measure that approximates the likelihood that any given possible output accurately models the real world. In effect, a lens gives structure to uncertainty about how an ETL process should interpret its input data.

Asking ETL designers to specify this structure manually for the entire ETL process is impractical. Lenses allow this structure to be specified as a composition of individual *simple* transformations, constraints, or target properties that take the place of normal operators in the ETL pipeline. However, composition requires closure. In this section, I define a closed framework for lens specification, and illustrate its generality through three example lenses.

3.1.1 The Lens Framework

A lens instance is defined over a query $Q(D)$, and is in turn responsible for constructing a PC-Table $(\mathbf{W}, P)$. A lens defines $\mathbf{W}$ as a C-Table through a VG-RA expression $\mathcal{F}_{lens}(Q(D))$. Independently, the lens constructs P as a joint probability distribution over every variable introduced by $\mathcal{F}_{lens}$, by defining a sampling process in the style of classical VG-functions [70], or supplementing it with additional meta-data to create a PIP-style grey-box [75]. An example of the complete process for the Domain Constraint Lens defined below is illustrated in Figure $3-1$. These semantics are closed over PC-Tables. If $Q(D)$ is non-deterministic — that is, the lens' input is defined by a PC-Table $(Q(D), P_Q)$ — the lens' semantics are virtually unchanged. VG-RA is closed over C-Tables, so $\mathcal{F}_{lens}(Q(D))$ simply defines a new C-Table. Defining P as an extension of P_Q with distributions for all variables newly introduced by $\mathcal{F}_{lens}$ provides closure for the probability measure, a topic I will return to in Section 3.1.3.

3.1.2 Lens Examples

I first illustrate the generality of the lens framework through three example lenses: domain constraint repair, schema matching, and archival. To construct a lens over query Q, the user writes:

```
CREATE LENS <lens_name> AS Q WITH  <lens_type>(<lens_arguments>);
```

Domain Constraint Repair. A domain constraint repair lens enforces attribute-level constraints such as `NOT NULL`. Under the assumption that constraint violations are a consequence of data-entry errors or missing values, domain constraint violations can be repaired by finding a legitimate replacement for each invalid value. Obtaining reliable replacement values typically requires detailed domain knowledge. However, in an on-demand setting, approximations are sufficient. The domain constraint repair

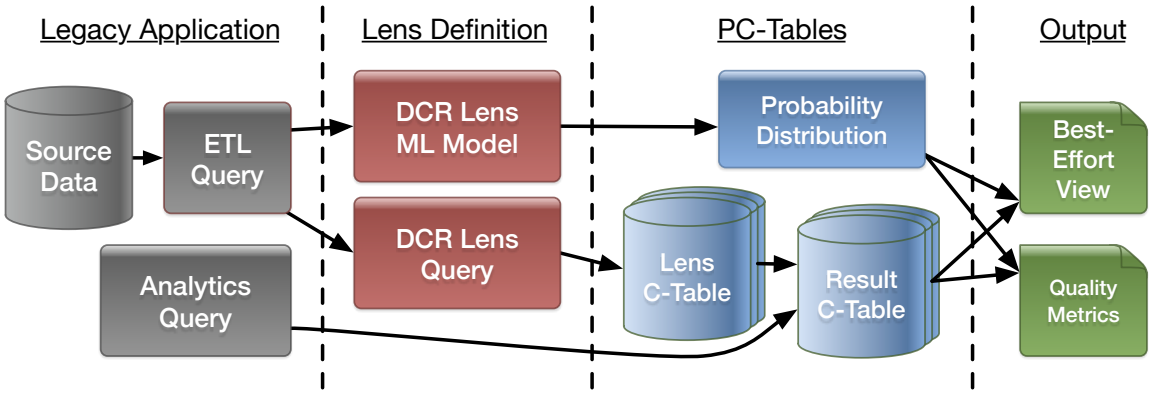


Figure 3-1: Example of the domain constraint repair lens applied in a legacy application. The output layer is discussed in Section 3.3.

lens uses educated guesses about data domains (e.g., uniform distributions over allowable values) and machine learning models (e.g., a naive Bayes classifier trained over $Q(D)$) to approximate domain knowledge. With $sch(Q) = \{\langle\, a_1, t_1 \,\rangle, \ldots, \langle\, a_n, t_n \,\rangle\}$ denoting the attributes a_i of $Q(D)$ and their type t_i, a domain constraint repair lens definition has the form:

$$\texttt{... WITH DOMAIN\_REPAIR(}a_1\ t_1\texttt{, ...,}\ a_n\ t_n\texttt{)}$$

The C-Table for the lens' output is constructed by the query $\mathcal{F}_{lens} = \pi_{\{\ldots,a_i \leftarrow V_i,\ldots\}}$, where each V_i is defined as:

$$\texttt{if}\ t_i \models a_i\ \texttt{then}\ a_i\ \texttt{else}\ Var(Name_i, \texttt{ROWID}))$$

In this expression, $t_i \models a_i$ if a_i satisfies the type constraints of t_i, and each $Name_i$ is a freshly allocated variable name. Independently, P is defined by training a classifier or similar model for each attribute on the output of Q.

Example 4 *Returning to Example 1, Alice creates a lens to handle missing values in the Product table:*

```
CREATE LENS SaneProduct AS SELECT * FROM Product
```

id	name	brand	category
P123	Apple 6s, White	$Var('X', \texttt{R1})$	phone
P124	Apple 5s, Black	$Var('X', \texttt{R2})$	phone
P125	Samsung Note2	Samsung	phone
P2345	Sony 60 inches	$Var('X', \texttt{R4})$	$Var('Y', \texttt{R4})$
P34234	Dell, Intel 4 core	Dell	laptop
P34235	HP, AMD 2 core	HP	laptop

Figure 3-2: The C-Table for `SaneProduct`

```
WITH DOMAIN_REPAIR( category string NOT NULL,
                    brand    string NOT NULL );
```

From Alice's perspective, the lens ***`SaneProduct`*** *behaves as a standard database view. However, the content of the lens is guaranteed to satisfy the domain constraints on* ***`category`*** *and* ***`brand`****.* ***`NULL`*** *values in these columns are replaced according to a classifier built over the output of the query over* ***`Product`****. Figure 3-2 shows the C-Table for this lens.*

Schema Matching. A schema matching lens creates a mapping from the source data's schema to a user-defined target schema. This is especially important for non-relational data like JSON objects or web tables, which may not have well defined schemas [66,71]. Given a destination schema $\{\langle\, b_1, t_1 \,\rangle, \ldots, \langle\, b_m, t_m \,\rangle\}$ and a threshold ω, a schema matching lens definition has the form:

$$\ldots \texttt{ WITH SCHEMA\_MATCHING(}b_1\ t_1\texttt{, }\ldots\texttt{, }b_m\ t_m\texttt{, }\omega\texttt{)}$$

The schema matching lens defines a fresh boolean variable $Var(Name_{i,j})$ for every pair a_i, b_j, where $\langle\, a_i, t_i \,\rangle \in sch(Q)$. The probability of $Var(Name_{i,j})$ corresponds to the probability of a match between a_i and b_j. $\mathcal{F}_{lens}$ takes the form: $\pi_{\{\ldots, b_j \leftarrow V_j, \ldots\}}$, where V_j enumerates possible matches for b_j:

if $Var(Name_{1,j})$ then a_1 else

if $Var(Name_{2,j})$ then a_2 else

if $Var(Name_{3,j})$ then a_3 else

$\vdots$

if $Var(Name_{n,j})$ then a_n else NULL

As an optimization, matches for type-incompatible pairs of attributes are skipped. Additionally, the lens discards matches where the likelihood of a schema-level match falls below a user-defined threshold (ω).

Example 5 *Alice next turns to the ratings data sets, which have incompatible schemas. She creates a lens and a joint view:*

```
CREATE LENS MatchedRatings2 AS SELECT * FROM Ratings2
  WITH SCHEMA_MATCHING( pid string, ..., rating float,
                        review_ct float, NO LIMIT );
CREATE VIEW AllRatings AS SELECT * FROM MatchedRatings2
                    UNION SELECT * FROM Ratings1;
```

The resulting C-Table for ***MatchedRatings2*** *is shown in Figure 3-3. From Alice's perspective,* ***AllRatings*** *behaves as a normal view combining* ***Ratings1*** *and* ***Ratings2****. Behind the scenes, the attributes of* ***Ratings2*** *are quietly matched against those of* ***Ratings1****. In this example, only* ***evaluation*** *and* ***num_ratings*** *are type compatible match candidates, and other match cases are dropped.*

Numerous options are available for constructing P, including domain-based schemes or complex ontology-based matching. However, even a simple matching scheme can

pid	...	rating	review_ct
P125	...	if $Var('rat = eval')$ then 3 else if $Var('rat = num\_r')$ then 121 else NULL	if $Var('rev\_ct = eval')$ then 3 else if $Var('rev\_ct = num\_r')$ then 121 else NULL
P34234	...	if $Var('rat = eval')$ then 5 else if $Var('rat = num\_r')$ then 5 else NULL	if $Var('rev\_ct = eval')$ then 5 else if $Var('rev\_ct = num\_r')$ then 5 else NULL
P34235	...	if $Var('rat = eval')$ then 4.5 else if $Var('rat = num\_r')$ then 4 else NULL	if $Var('rev\_ct = eval')$ then 4.5 else if $Var('rev\_ct = num\_r')$ then 4 else NULL

Figure 3-3: The C-Table for `MatchedRatings2`

be sufficient for On-Demand ETL. I approximate the probability of a match between two attributes by a normalized edit distance between the two attribute names. As I show in Section 5.4 (Figure 3-13), even this simple matcher can produce suitable results.

Archival. An archival lens captures the potential for errors arising from OLAP queries being posed over stale data [78], like queries run in between periodic OLTP to OLAP bulk data copies. The lens takes a list of pairs $\langle\, T, R \,\rangle$, where R is a reference to a relation in an OLTP database, and T is the period with which R is locally archived.

$$\ldots \texttt{ WITH ARCHIVAL(} \langle\, T_1, R_1 \,\rangle, \ldots, \langle\, T_m, R_m \,\rangle \texttt{)}$$

This lens probabilistically discards rows from its output that are no-longer valid according to the lens query $\mathcal{F}_{lens} = \sigma_{Var(Name,\texttt{ROWID})}$, where $Name$ is a freshly allocated identifier. In the background, the lens periodically polls for samples drawn from each R_j to estimate the volatility of each relation referenced by Q. Denote by ν_j the probability of a tuple in R_j being invalidated at some point during the period T_j. P is defined independently for each row as a binomial distribution with probability $\prod_{\{j|R_j \in Q\}} \nu_j$.

3.1.3 Composing Lenses

Example 6 *When Alice examines* `AllRatings`*, she suddenly realizes that the data in* `Ratings1` *is missing* `rating` *information. She creates a domain repair lens:*

```
CREATE LENS SaneRatings AS
SELECT pid, category, rating, review_ct FROM AllRatings
WITH DOMAIN_REPAIR(rating DECIMAL NOT NULL)
```

The C-Table for `SaneRatings` *is straightforward to construct, as both lenses involved can be expressed as VG-RA expressions. However, the domain repair lens must still train a model to fill in distributions for missing values. In contrast to Example 4, where the model was trained on deterministic input, here the input is a PC-Table.*

The closure of VG-RA over PC-Tables requires that any non-deterministic query $\mathcal{F}$ be defined alongside a process that extends the input PC-Table's probability measure P_{in} to cover any variables introduced by $\mathcal{F}$. For lenses, there are three possibilities. In the trivial case where $\mathcal{F}$ introduces no new variables, P_{in} remains unmodified. In the second case, variables introduced by $\mathcal{F}$ are independent of P_{in} and a joint distribution is defined trivially as the product of the original and new distributions. If any new variables depend on P_{in}, a grey-box distribution definition [75] can be used to express these dependencies directly.

However, it may not always be possible to explicitly define dependencies, particularly when adapting existing on-demand cleaning solutions. Mimir provides three separate mechanisms to enable support for lenses that require deterministic inputs: (i) Train the lens on the most-likely output of the source lens (see Section 3.3), (ii) Train the lens on samples of rows drawn from random instances of the source model, or (iii) Train the lens on the subset of the source data that is fully deterministic (i.e., certain).

The above lens combining methods behave like a pipeline, where one lens is based on the result of the other. This method is error-prone, the error can propogate from one lens to the other. Instead of combining lenses in a pipelined fashion like

Example 6, an alternate method is to consider all the lenses as a whole and construct a probabilistic graphical model or a markov logic network to express them. Then inference can then be used on these models or networks to calculate P_{in} in a uniform way.

Example 7 *Alice issues the following query:*

```
SELECT p.pid, p.category, r.rating, r.review_ct
FROM SaneRatings r NATURAL JOIN Product p
WHERE p.category IN ('phone','TV') OR r.rating > 4
```

The resulting C-Table is shown in Figure 3-4. The first two products are entirely deterministic. P125 is a phone and deterministically satisfies the query, but has attribute-level uncertainty from schema matching (Example 5). P2345 has a missing `category` *(Example 4) and* `rating` *(Example 6), so the row's condition is effectively the entire selection predicate. P34234 and P34235 are laptops and fail the test on* `category`*, so their presence in the result set depends entirely on how* `rating` *is matched (Example 5). Recall that there are three candidates:* `evaluation`*,* `num_ratings`*, or neither. In the last case, domain repair (Example 6) replaces the* `NULL` *with* $Var('Z', R11)$ *and* $Var('Z', R12)$*. P34234 and P34235 have functional* `if` *expressions in their conditions, with the form* $(\texttt{if}\ \phi_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3) > 4$*. These expressions can be simplified by recursively pushing the comparison into the branches:* $\texttt{if}\ \phi_1\ \texttt{then}\ e_2 > 4\ \texttt{else}\ e_3 > 4$*, in-lining the branches into the condition:* $(\phi_1 \wedge (e_2 > 4)) \vee (\neg\phi_1 \wedge (e_3 > 4))$*, and then further simplifying the resulting boolean expression.*

id	category	rating	review_ct	ϕ (condition)
P123	phone	4.5	50	$\top$
P124	phone	4	100	$\top$
P125	phone	if $Var('rat = eval')$ then ... ...else $Var('Z', R10)$	if $Var('rat = eval')$ then ... ...else NULL	$\top$
P2345	$Var('Y', R4)$	$Var('Z', R8)$	245	$(Var('Y', R4) = 'phone')$ $\vee (Var('Y', R4) = 'TV')$ $\vee Var('Z', R8) > 4$
P34234	laptop	if $Var('rat = eval')$ then ... ...else $Var('Z', R11)$	if $Var('rat = eval')$ then ... ...else NULL	$Var('rat = eval')$ $\vee Var('rat = num\_r')$ $\vee (Var('Z', R11) > 4)$
P34235	laptop	if $Var('rat = eval')$ then ... ...else $Var('Z', R12)$	if $Var('rat = eval')$ then ... ...else NULL	$Var('rat = eval')$ $\vee(\neg Var('rat = num\_r')$ $\wedge (Var('Z', R12) > 4))$

Figure 3-4: C-Table for the query over `SaneRatings` and `SaneProduct`

3.2 Probabilistic Query Processing

I next address the challenge of deploying the PQP techniques necessary to support Lenses into an existing database or ETL pipeline. My approach, called *Virtual C-Tables* or VC-Tables, works by decomposing VG-RA queries into deterministic and non-deterministic components. Non-deterministic components are percolated out of queries, making it possible for the bulk of the ETL process to remain within a classical deterministic system. A small On-Demand ETL shim layer wraps around the database, and provides a minimally-invasive interface for uncertainty-aware users and applications. This shim layer is also responsible for managing several views, discussed in Section 3.3, that provide backwards compatibility for legacy applications.

Let $\mathcal{F}(D)$ denote a VG-RA query over a deterministic database D. When combined with a probability measure P, $(\mathcal{F}(D), P)$ defines a PC-Table. Semantics for deterministic queries over PC-Tables are well defined, but rely on support for labeled nulls, a feature not commonly found in popular data management systems. Existing probabilistic query processing systems address this limitation by restricting themselves to special cases like finite-discrete probability distributions over categorical data [50, 67], relying on costly user-defined types [70, 75, 77], or by specializing the entire database for uncertain data management [9, 54, 111]. Ultimately, each of these

solutions is either too specialized for Mimir, or too disruptive to be deployed into an existing classical ETL pipeline or databases.

Virtual C-Tables decouple the deterministic components of a query from the non-deterministic components that define a PC-Table. This decomposition is enabled by the observation that once the probability measure P of a PC-Table $(\mathcal{F}(D), P)$ is constructed, further *deterministic* queries Q over the PC-Table do not affect P. Consequently, I am free to rewrite the C-Table $Q(\mathcal{F}(D))$ defined by any query over $(\mathcal{F}(D), P)$ into any *equivalent* query $\mathcal{F}'(Q'(D))$, where Q' is deterministic and $\mathcal{F}'$ is non-deterministic. In this new, normalized form, the heavy-weight deterministic inner query Q' can be evaluated by a traditional database, while a much simpler $\mathcal{F}'$ can be evaluated or analyzed by a small shim layer sitting between the database and its users. Further queries $q(\mathcal{F}'(Q'(D)))$ can likewise be rewritten into normal form $\mathcal{F}''(q'(Q'(D)))$, enabling views and `SELECT INTO` queries, both of which frequently appear in ETL workflows.

3.2.1 Normal Form VG-RA

Non-determinism arises in VG-RA queries through expressions containing variable terms — that is, only through projection and selection. Correspondingly, I propose a normal form of VG-RA: $\pi_{a_i \leftarrow e_i}(\sigma_\phi(Q(D)))$, where the source query $Q(D)$ is expressible using classical bag-relational algebra. The two outer operators, which I represent jointly as $\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)$, fully encode the branching possibilities of the C-Table. Figure 3-5 shows how any query in VG-RA can be rewritten into this form by percolating all expressions with a VG-Term $Var(\ldots)$ up through the relational algebra tree.

Projection and selection operators wrapping around $\mathcal{F}$ may be in-lined into $\mathcal{F}$

$$\pi_{a'_j \leftarrow e'_j}\left(\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)(Q(D))\right) \equiv \mathcal{F}(\langle\!\langle\, a'_j \leftarrow [[e'_j(a_i \leftarrow e_i)]]_{lazy} \,\rangle\!\rangle, \phi)(Q(D)) \quad (3.1)$$

$$\sigma_{\psi}\left(\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)(Q(D))\right) \quad \equiv \quad \mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi \ \wedge \ \psi_{var})(\sigma_{\psi_{det}}(Q(D))) \quad (3.2)$$

$$\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)(Q(D)) \times \mathcal{F}(\langle\!\langle\, a'_j \leftarrow e'_j \,\rangle\!\rangle, \phi')(Q'(D)) \\ \equiv \mathcal{F}(\langle\!\langle\, a_i \leftarrow e_i, a'_j \leftarrow e'_j \,\rangle\!\rangle, \phi \wedge \phi')(Q(D) \times Q'(D)) \quad (3.3)$$

$$\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)(Q(D)) \uplus \mathcal{F}(\langle\, a_i \leftarrow e'_i \,\rangle, \phi')(Q'(D)) \\ \equiv \mathcal{F}(\langle\, a_i \leftarrow [[\texttt{if } src = 1 \texttt{ then } e_i \texttt{ else } e'_i]]_{lazy} \,\rangle, \\ [[\texttt{if } src = 1 \texttt{ then } \phi \texttt{ else } \phi']]_{lazy})(\\ \pi_{*,src \leftarrow 1}(Q(D)) \uplus \pi_{*,src \leftarrow 2}(Q'(D))) \quad (3.4)$$

Figure 3-5: Recursive reduction to Normal Form.

according to rewrites 3.1 and 3.2. As an optimization, expressions and conditions are simplified through lazy evaluation, and predicates ψ are partitioned into two components: $\psi_{var} \wedge \psi_{det} \equiv [[\psi(a_i \leftarrow e_i)]]_{lazy}$, having and not having variable terms, respectively.

Cross-products of two normalized expressions are composed in the straightforward way by concatenating attribute sets and conjunctively combining local conditions as shown in rewrite 3.3. We use alpha-renaming to avoid schema conflicts between $Q(D)$ and $Q'(D)$, and without loss of generality, assume that the intersection of a_i and a'_j is empty.

Finally, rewrite 3.4 shows how bag-unions can be rewritten by injecting a provenance marker into the deterministic queries. A fresh attribute src distinguishes rows originating from each of two source queries Q and Q'.

3.2.2 Virtual Views

Normalization allows lenses to be incorporated into existing ETL pipelines. A lens constructs a probability measure P out of its input $Q(D)$, and a C-Table using $\mathcal{F}_{lens}(Q(D))$. The lens query and any subsequent queries over it are normalized into a normal form query $\mathcal{F}'(Q'(D))$, and the view $Q'(D)$ is constructed and materialized by the traditional database. $\mathcal{F}'$ is stored alongside Q' and defines a *virtual view* for $\mathcal{F}'(Q'(D))$. The shim interface transparently normalizes queries over virtual views $q(\mathcal{F}'(Q'(D)))$ to $\mathcal{F}''(q'(Q'(D)))$, allowing $q'(Q'(D))$ to be evaluated by the traditional database. View definitions and `SELECT INTO` queries are similarly rewritten, defining new virtual views instead of their normal behavior.

3.2.3 Partition

In Section 3.2.1, I showed the rules for recursive reduction to normal form. These rules are very general, but the resulting queries are not efficient. For example, for join operations, the rule generalized it to cross product, which increases the complexity tremendously.

Most existing provenance-based evaluation strategies [9,17,28,50,111] limit themselves to supporting finite, discrete data distributions. This is a necessary concession to efficiency, as non-deterministic joins over data drawn from a continuous distribution effectively devolve to cross products. To avoid completely devolving to cross-product performance, my partition query evaluation strategy is based on the assumption that a comparatively small fraction of the user's data is uncertain. Therefore, instead of generalizing the operators which will be applied to all deterministic and non-deterministic data, only non-deterministic data is selected to be reduced to normal form. I first rewrite to query into several subqueries, where the result of each subquery shares the

same provenance. Only those subqueries with non-deterministic result will reduced to normal form. In this way, the backend database is fully utilized and the extra complexity introduced by reduction process is minimized.

In this section, in the context of a specific subquery, I will use ϕ_i to represent a boolean expression that captures the lineage of attributes and rows in the subquery. I use ψ_i to denote the where clause for the sub-query. To partition a query ($Q(D)$), I begin with a set of partitions, each defined by a boolean formula ψ_i over attributes in $sch(Q)$. For each partition ψ_i I can simplify the selection condition ϕ of a query Q into a reduced form ϕ_i. I use $\phi[\psi_i]$ to denote the result of propagating the implications of ψ_i on ϕ. For example, (**if** X `is null` **then** $Var('X', ROWID)$ **else** X)$[X$ `is null`$] \equiv Var('X')$. For a set of partitions to be used to split a query into fragments it must be complete ($\bigvee \psi_i \equiv T$) and disjoint ($\forall i \neq j \;.\; \phi[\psi_i] \rightarrow \neg\phi[\psi_j]$).

Given a set of partitions $\Psi = \{\psi_1, \ldots, \psi_N\}$, the partition rewrite transforms the original query into an equivalent set of partitioned queries as follows:

$$(\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)(Q(D))) \mapsto \mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi_{var,1})(\sigma_{\psi_1 \wedge \phi_{det,1}}(Q(D))) \\ \cup \cdots \cup \mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi_{var,N})(\sigma_{\psi_N \wedge \phi_{det,N}}(Q(D)))$$

where $\phi_{var,i}$ and $\phi_{det,i}$ are respectively the non-deterministic and deterministic conditions of ϕ (i.e., $\phi = \phi_{var,i} \wedge \phi_{det,i}$) for each partition. Partitioning then, consists of two stages: (1) Obtaining a set of potential partitions Ψ from the original condition ϕ, and (2) Segmenting ϕ into a deterministic filtering predicate and a non-deterministic lineage component.

Algorithm 1 $\texttt{isDet}(E)$

Require: E: An arithmetic expression that may contain Var terms.
Ensure: An expression that is true when E is deterministic.

```
if E ∈ {ℝ, ⊤, ⊥} then
  return ⊤
else if E is Var then
  return ⊥
else if E is Column_i then
  return D_i
else if E is ¬E_1 then
  return isDet(E_1)
else if E is E_1 ∨ E_2 then
  return (E_1 ∧ isDet(E_1)) ∨ (E_2 ∧ isDet(E_2))
              ∨ (isDet(E_1) ∧ isDet(E_2))
else if E is E_1 ∧ E_2 then
  return (¬E_1 ∧ isDet(E_1)) ∨ (¬E_2 ∧ isDet(E_2))
              ∨ (isDet(E_1) ∧ isDet(E_2))
else if E is E_1 {+, −, ×, ÷, =, ≠, >, ≥, <, ≤} E_2 then
  return (isDet(E_1) ∧ isDet(E_2))
else if E is if E_1 then E_2 else E_3 then
  return isDet(E_1) ∧ (   (E_1 ∧ isDet(E_2))
                        ∨(¬E_1 ∧ isDet(E_3))   )
```

Partitioning the Query

Algorithm 2 begins with the selection predicate ϕ in the shim query $\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)$, and outputs a set of fragments $\Psi = \{\psi_i\}$. Fragments are formed from the set of all possible truth assignments to a set of candidate conditions. Candidate conditions are obtained from if statements appearing in ϕ that have deterministic conditions, and that branch between deterministic and non-deterministic cases.

Example 8 *Recall in Example 4, I generate a new C-Table using the VG-RA query. I now issue a query:*

```
SELECT type FROM SaneProduct
WHERE brand = 'Apple' AND category = 'phone'
```

Algorithm 2 `naivePartition`(ϕ)

Require: ϕ: A non-deterministic boolean expression
Ensure: Ψ: A set of partition conditions $\{\psi_i\}$
  conditions $\leftarrow \emptyset$
  $\Psi \leftarrow \emptyset$
  for (**if** *condition* **then** α **else** β) $\in$ `subexps`(ϕ) **do**
    /* Check `if`s in ϕ for candidate partition conditions */
    if `isDet`(*condition*) $\wedge$ (`isDet`(α) $\neq$ `isDet`(β)) **then**
      conditions $\leftarrow$ *conditions* $\cup \{$*condition*$\}$
  /* Loop over the power-set of conditions */
  for *partition* $\in 2^{conditions}$ **do**
    $\psi_i \leftarrow \top$
    /* Conditions in the partition are true, others are false */
    for *clause* $\in$ *conditions* **do**
      if *clause* $\in$ *partition* **then** $\psi_i \leftarrow \psi_i \wedge$ *clause*
                else $\psi_i \leftarrow \psi_i \wedge \neg$*clause*
    $\Psi \leftarrow \Psi \cup \{\psi_i\}$

The query has the non-deterministic condition (ϕ):

$$(\textbf{if } brand \texttt{ is null } \textbf{then } Var('b', \texttt{ROWID}) \textbf{ else } brand) = \text{`}Apple'$$
$$\wedge\ (\textbf{if } cat \texttt{ is null } \textbf{then } Var('c', \texttt{ROWID}) \textbf{ else } cat) = \text{`}phone'$$

There are two candidate conditions in ϕ: brand `is null` *and cat* `is null`*. Thus, Algorithm 2 creates 4 partitions:* $\psi_1 = (\neg brand$ `is null` $\wedge \neg cat$ `is null`$)$, $\psi_2 = (brand$ `is null` $\wedge \neg cat$ `is null`$)$, $\psi_3 = (\neg brand$ `is null` $\wedge\ cat$ `is null`$)$, *and finally* $\psi_4 = (brand$ `is null` $\wedge\ cat$ `is null`$)$.

Segmenting ϕ

Using `isDet` from Algorithm 1, I partition the conjunctive terms of $\phi[\psi_i]$ into deterministic and non-deterministic components $\phi_{i,det}$ and $\phi_{i,var}$, respectively so that

$$(\phi_{i,det} \wedge \phi_{i,var}) \equiv \phi[\psi_i]$$

Note that Algorithm 2 may return a set of conditions that is not disjoint. I apply an additional check for overlap before using the output of this algorithm to partition a query.

Partitioning Complex Boolean Formulas

I next describe a more aggressive partitioning strategy that uses the structure of ϕ to create partitions where each partition depends on exactly the same set of Var terms. To determine the set of partitions for each sub-query, I use a recursive traversal through the structure of ϕ, as shown in Algorithm 3. The idea of the algorithm is that, in a fine-grained partition, there are exactly 2^N sub-queries union-ed together, where N is the number of atoms in where clause. For each subsets i (i from 1 to 2^N) of atoms, Algorithm 3 generates the condition ϕ_i and the corresponding selection predicate to select all rows having the same lineage.

Algorithm 3 generalPartition

Require: ϕ: A non-deterministic boolean expression considered as a tree structure, a set of atoms $\{a_i\}$
Ensure: Ψ: A set of partitions $\{\psi_i\}$ and corresponding conditions $\{\phi_i\}$
  if ϕ is a single atom **then**
    return
  if ϕ.leftChild is an operator **then**
    generalPartition(root.leftchild)
  if ϕ.rightChild is an operator **then**
    generalPartition(root.rightchild)
  constructPartition(ϕ,$\{a_i\}$);

The partition approach makes full use of the backend database engine by splitting the query into deterministic and non-deterministic fragments. The lineage of the condition for each sub-query is simpler, and typically no longer data-dependent. As a consequence, explanation objects can be shared across all rows in the partition. The number of partitions obtained with both partitioning schemes is exponential in the

Algorithm 4 constructPartition

Require: ϕ: A non-deterministic boolean expression considered as a tree structure,$\{a_i\}$
Ensure: Ψ: A set of partitions $\{\psi_i\}$ and corresponding conditions $\{\phi_i\}$

```
if {a_i} contains φ.leftChild.φ_i and
{a_i} contains φ.rightChild.φ_i then
  φ_i.combine(φ.leftChild.φ_i,φ.rightChild.φ_i);
  ψ_i.add(φ.leftChild.ψ_i);
  ψ_i.add(φ.rightChild.ψ_i);
else
  if {a_i} contains φ.leftChild.φ_i then
    φ_i.add(φ.leftChild.φ_i);
    if φ is instanceOf OR Operator then
      ψ_i.add(NOT φ.rightChild);
    if φ is instanceOf AND Operator then
      ψ_i.add(φ.rightChild);
  else
    if {a_i} contains φ.rightChild.φ_i then
      φ_i.add(φ.rightChild.φ_i);
      if φ is instanceOf OR Operator then
        ψ_i.add(NOT φ.leftChild);
      if φ is instanceOf AND Operator then
        ψ_i.add(φ.leftChild);
    else
      ψ_i.add(φ);
```

number of candidate conditions. Partitions could conceivably be combined, increasing the number of redundant tuples processed by Mimir to create a lower-complexity query. In the extreme, we might have only two partitions: one deterministic and one non-deterministic.

3.3 Result Quality Analysis

Using virtual views, queries over lens outputs are rewritten into the normal form $\mathcal{F}(Q(D))$, and $Q(D)$ is evaluated by the database. However, the C-Table construction query $\mathcal{F}$ is of minimal use in its raw form. I next turn to the construction of user-consumable summaries of $\mathcal{F}$.

3.3.1 Summarizing the Result Relation

Users consume a Virtual C-Table $\mathcal{F}(\langle\, a_i \leftarrow e_i \,\rangle, \phi)(Q(D))$ through one of two deterministic summary relations: A deterministic relation $\mathcal{R}_{det}$, and a best-guess relation $\mathcal{R}_{guess}$. The deterministic relation $\mathcal{R}_{det}$ represents the certain answers [49] of the virtual C-Table, and is constructed by replacing every variable reference in each e_i and ϕ with `NULL`, and dropping rows where $\phi \neq \top$:

$$\texttt{SELECT } e_i(* \to \texttt{NULL}) \texttt{ AS } a_i \texttt{ FROM } Q(D) \texttt{ WHERE } \phi(* \to \texttt{NULL})$$

The resulting relation contains all of the rows deterministically present in $\mathcal{F}(Q(D))$, with `NULL` taking the place of any non-deterministic values. $\mathcal{R}_{det}$ can be computed entirely within a classical deterministic database, making it backwards compatible with legacy ETL components.

The best-guess relation $\mathcal{R}_{guess}$ is constructed in two stages. First, the deterministic database system executes $Q(D)$. As the classical database streams results for $Q(D)$, the shim layer evaluates each e_i and ϕ based on the valuation given by $\texttt{argmax}_v(P(v))$, the most-likely possible world. Field values or row confidences in the best guess relation that depend on v are annotated in the shim layer's output. Legacy applications can quietly ignore this annotation. In uncertainty-aware applications, this annotation is used to indicate which parts of the result are uncertain to the end-user.

Example 9 *Continuing Example 7, the database now responds to Alice's query with the most-likely result shown in Figure 3-6. Every non-deterministic (i.e., guessed) field is annotated with an asterisk. Every row with a non-deterministic condition is similarly marked. A footer indicates how many rows were dropped due to a non-deterministic condition evaluating to false in the most likely possible world. Note that this best-guess estimate is not entirely accurate:* `evaluation` *has been mapped to* `review_ct`*, and* `rating` *has not been matched, resulting in best-effort guesses of 2*

	id	category	rating	review_ct	
	P123	phone	4.5	50	
	P124	phone	4	100	
	P125	phone	2 *	3 *	
	P34235	laptop	5 *	4.5 *	*

(Up to 2 results may be missing. *)

Figure 3-6: The best-guess summary of the C-Table from Figure 3-4 that Alice actually sees.

and 5 for the last two rows of the result. In spite of the error, Alice can quickly see the role uncertainty plays in her results.

3.3.2 Summarizing Result Quality

Once the user is made aware that parts of a query result may be uncertain, two questions likely to be asked are "How bad?" and "Why?". Answering the latter question is trivial: $\mathcal{F}$ contains a reference to all of the variables that introduce uncertainty, each of which is explicitly linked to the lens that constructed it. In other words, $\mathcal{F}$ serves as a form of provenance that can be used to explain sources of uncertainty to the end-user.

The former question requires us to develop a notion of result quality. Our approach is based on the idea of *noise*: intuitively, the less noise is present in the model, the higher the quality of the best-guess relation's predictions. I abstractly define result quality as the level of confidence that the user should have in the annotated best-guess results. These results include both non-deterministic attribute values, as well as possible tuples.

Recall that a non-deterministic value in $\mathcal{R}_{guess}$ is obtained from non-deterministic expressions in $\mathcal{R}_{guess}$. Numerous metrics that effectively convey the quality of attribute values drawn from a probabilistic database have been proposed, including pessimistic hard bounds [78], variance [70, 75], and $\epsilon - \delta$ bounds [64].

As a simplification, I assume that the cognitive burden of understanding uncertainty in a specific attribute value is constant, while the burden of tracking the presence or absence of rows in the output scales linearly. Intuitively, guessing wrong about tuple presence can mean the difference between overwhelming the user with a flood of unnecessary results, and hiding the presence of potentially critical information. Under this assumption, tuple-level uncertainty adds more noise to the result, and I will focus primarily on this type of uncertainty from here on.

The appearance of a tuple t in a query result is determined by the ground truth of its local condition $t.\phi$. Valuations $v(\Sigma)$ map $t.\phi$ to a deterministic boolean value $t.\phi[v]$. From the PC-Table's probability measure $P(v)$, I get the binomial distribution $P(t.\phi[v])$, often called the confidence of t. I use the confidence of t to measure how difficult it is for the analyst to predict the ground truth of $t.\phi$. Intuitively, if $P(t.\phi[v])$ is skewed towards 0 or 1, I expect to predict the value of $t.\phi$ with reasonable accuracy; on the other hand, if $P(t.\phi[v])$ is a fair coin flip, I have no reliable information about the expected result of $t.\phi$. It is natural to use Shannon entropy as a metric to quantify the quality of the query result. I define the entropy of a tuple in terms of its confidence $p_t = P(t.\phi[v])$ as:

$$entropy(t) = -(\ p_t \cdot \log_2(p_t) + (1 - p_t) \cdot \log_2(1 - p_t)\ )$$

Efficiently approximating tuple confidences by sampling from $P(v)$ is well studied in probabilistic databases [67, 95], and I use similar techniques for estimating tuple entropies.

To unify the individual per-attribute and per-row metrics, I define a relation-wise noise function $\mathcal{N}(\mathcal{R})$ as a linear combination of individual metrics. For example, a relation $\mathcal{R}$ without non-deterministic attributes might have $\mathcal{N}(\mathcal{R}) = \sum_{t \in \mathcal{R}} entropy(t)$.

To account for the entropy generated by non-deterministic attributes, I start with the intuition that each attribute in the output provides $\frac{1}{N}$th of the information content of a tuple, where N is the arity of $\mathcal{R}$. Thus, by default, I assume each non-deterministic value contributes to the noise seen in the final result a fraction in the range $[0, \frac{1}{N}]$ inversely proportional to the attribute's estimated variance.

3.4 Pay-as-you-go Data Cleaning

When the analyst is given a query result $\mathcal{R}$ that does not meet her quality expectations, she can allocate additional resources for gathering more evidence. For example, she may spend some time gathering ground-truth values for variables in the output C-Table. By construction, variables represent uncertainty about basic facts. For example, a schema matching lens generates expressions of the form $Var('rat = eval')$ that could be stated as a simple question like "*Do* `rating` *and* `evaluation` *mean the same thing?*". Replacing variables with their ground truths means performing these basic curation tasks, with the goal of reducing the noise seen in the final result. Since $\mathcal{N}(\mathcal{R})$ depends on the entropy generated by the tuples in $\mathcal{R}$, in expectation, each curation task will reduce the noise seen in the final result. Identifying the variable that affects $\mathcal{N}(\mathcal{R})$ the most is not trivial: depending on $\{\ t.\phi \mid t \in \mathcal{R}\ \}$ some variables may generate more noise in the final result than others. In a perfect world, the analyst would simply replace variables in $\mathcal{R}$ until the required level of quality is reached. In the real world, curation tasks are expensive, and the optimal cleaning strategy depends on both quality goals and budget constraints. Hence, deciding a good strategy is essentially a resource allocation problem. I assume that a cost model is given by each lens in the form of a cost function $c(\cdot)$. This function maps variables to positive real numbers that represent the effort, or dollar cost of discovering the ground truth

for the given variable.

3.4.1 Prioritizing Curation Tasks

Prioritizing curation tasks is a dynamic decision process, as the outcome of one curation task affects the choice of the next one to be performed. Let's assume, for the moment, that the analyst has no budget constraints and her goal is simply to determine the ground truth of a given condition formula ϕ, minimizing the *expected* amount of resources spent in the process. In the literature, this optimization problem is known as *stochastic boolean function evaluation* [39, 116]. Both exact and approximated algorithms have been proposed for several classes of formulas. In its general form, the problem can be thought of as a *Markov Decision Process* having one state for each partial assignment to the variables in ϕ and one action for each variable (a curation task). Rewards are determined by $-c(\cdot)$ and state-transitions are determined by $P(v)$. Final states consist of assignments that make ϕ either true or false with certainty. The planning horizon is finite and equal to the number of variables in ϕ. A simple solution to the problem consists of a *policy*, prescribing a curation task for each non-terminating assignment to perform. The application of a policy is an interactive process: the system instructs the analyst to address a particular curation task ("*Do* `rating` *and* `evaluation` *mean the same thing?*"), the analyst provides the required ground truth, and asks the system for the next move. This feedback loop continues until the deterministic value of ϕ is obtained. As a baseline for evaluation, On-Demand ETL implements a naïve algorithm for computing policies of this kind, named naïve minimum expected total cost (NMETC).

Naïve Minimum Expected Total Cost (NMETC)

If a query result does not meet the analyst's standards, on-Demand ETL suggests ways to improve the quality by making better guesses of row confidences or non-deterministic attribute values. In order to obtain a higher quality of query result, we need to make a better guess of the truth value of ϕ_i. I incrementally explore user feedback to obtain more information of the variables to make a more certain guess. As a simplification I will focus on the problem of making possible-tuples more certain. Without loss of generality, I will treat each atom in the tuple's condition as a boolean variable. The resulting problem resembles stochastic boolean function evaluation [39], except that I model it as an interactive process: The database prioritizes curation tasks for the analyst, and the analyst responds by performing a task (i.e., by providing a ground truth value for one variable). The process repeats until the analyst is satisfied with the quality of the result. The probability distribution $\mathrm{p}(\phi[v])$ can also be viewed as a prior and incrementally reinforce the prior with user feedback as evidence.

Figure 3-7 illustrates a trace of this interaction for the confidence of one row tagged with the condition $\phi = A \vee (B \wedge C)$, visualized as a game tree. Each rectangle represents the result of the ϕ after applying user's immediate choice-point, while circles represent variables in the formula that define curation tasks. Each circle has two branches denoting two sets of possible worlds that correspond to the two possible ground truth values for the variable. Each circle node is annotated by the expected cumulative cost of reaching that point in the tree. The system and the user take turns to move from root towards a leaf representing a truth value of ϕ_i.

Clearly, there is a trade-off between the improvement of result quality and cost associated with it. Possible strategies are: pick a target utility and minimize the

cost to achieve the utility, or fix a total cost and maximize the utility to given the cost. In this section, I will show the strategies to play with the trade-off. In either scenario, Our problem becomes: given a set of expressions Φ, and the probability distributions of the variables, the cost of obtaining the truth values of the variables $C = \{c_1, c_2, \cdots, c_m\}$, Each path in the decision-tree from root to the a leaf represents a sequence of questions to the user and the leaf represents a guess. Notice that the correlation of expressions is constructed on common variables referenced from both expressions. We can group correlated expressions and compute the strategy for each expression group in parallel.

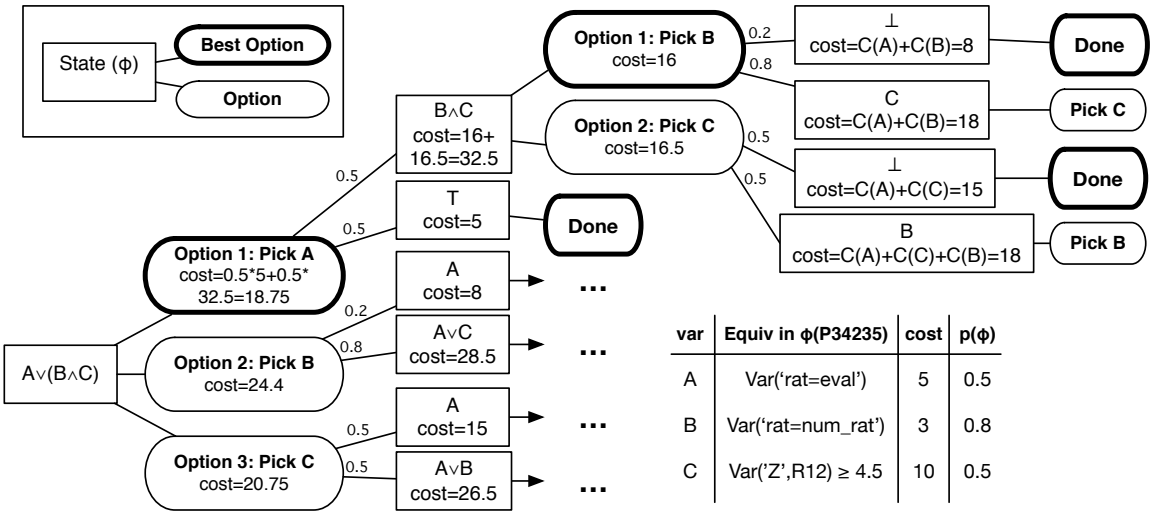


Figure 3-7: An example of the Naïve Minimum Expected Total Cost (NMETC).

3.4.2 Balancing Result Quality and Cost

Real-world ETL applications are unlikely to be free from budget constraints. Even when budget is not a problem, the average analyst will rarely aim for perfect information. Instead, she would rather target a reasonable approximation of the value of ϕ, setting an upper-bound on the entropy of the formula. Hence, I generalize the approach discussed above and make the assumption that the analyst wants to plan her

curation tasks so to maximize a hidden value function $\mathcal{V}(\cdot)$, which depends on $c(\cdot)$ and $\mathcal{N}(\cdot)$ and is unknown to the system. Clearly, I assume $\mathcal{V}(\cdot)$ decreases monotonically as the cumulative cost increases, and increases monotonically as the noise decreases. In simple words, $\mathcal{V}$ determines how much the analyst is willing to pay for an improvement in the estimation of the value of ϕ, on a case-by-case basis. I call this trade-off the *cost of perfect information* (CPI). Since the details of $\mathcal{V}(\cdot)$ are unknown, the goal of the system is to propose several candidate policies. Each policy should guarantee a certain expected entropy at the price of a certain expected cumulative cost. The user is then able to choose the candidate policy that best matches her hidden value function. Since the analyst may be subject to budget constraints hidden to the system, the candidate list includes greedy versions of the policies, computed progressively over limited planning horizons. Inspired by the algorithms EG2 [93], CS_ID3 [115] and CS_C4.5 [51], On-Demand ETL supports the following four greedy strategies:

Algorithm	$CPI(v_i)$
EG2	$(2^{IG[\mathcal{R}(v_i)]} - 1)/c_i$
CS_ID3	$(IG[\mathcal{R}(v_i)]^2)/c_i$
CS_IDX	$IG[\mathcal{R}(v_i)]/c_i$
CS_C4.5	$IG[\mathcal{R}(v_i)]/\sqrt{c_i}$

Here, IG denotes the *information gain*, or the reduction in noise produced by the curation task on variable v_i.

Example 10 *Consider the condition ϕ for P34235 in Figure 3-4, which has the form $A \vee (B \wedge C)$. Figure 3-8 illustrates the decision tree that ranks curation tasks (the three variables), given lens-defined ground-truth costs and marginal probabilities as shown in the figure. The expected entropy after performing the curation task for v_i, denoted by $E[H(v_i)]$, is computed as a weighted average over all possible outcomes of*

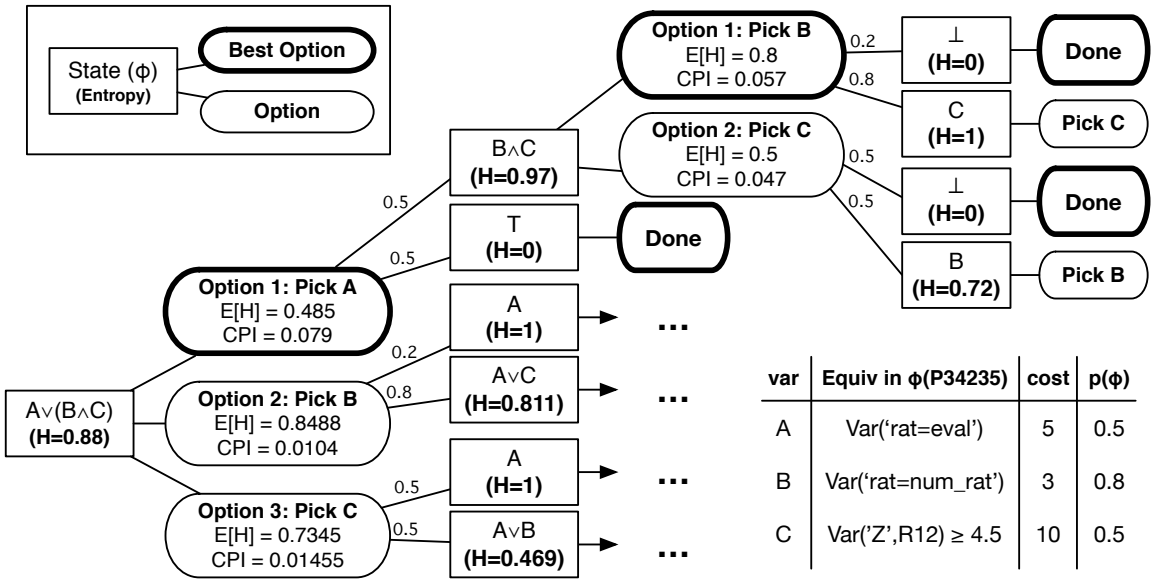


Figure 3-8: An example of the CS_IDX algorithm optimizing CPI.

the task. $CPI(v_i)$ *is computed according to the CS_IDX formula given above, with* $IG[\mathcal{R}(v_i)] = H - E[H(v_i)]$.

3.5 Experiments

In this section we show the feasibility of On-Demand ETL and explore several points in its design space. Specifically, we find that: (i) The greedy approach of minimizing CPI produces higher-quality query results at lower costs than optimizing for total cost when the hidden value function is not known, (ii) The precise formula used to compute CPI is not critical to achieving high quality results, (iii) When composing lenses, order is relevant, as open-ended lenses like domain-constraint repair can fix issues created by other lenses earlier in the pipeline, and (iv) Tree-based classifiers work best for domain constraint repair lenses.

3.5.1 Experimental Setup

Our experimental setup consists of three data sets drawn from publicly available sources. To simulate data-entry error, a portion of the data values are randomly removed. To simulate an analyst's querying behavior, we identify one attribute in each data set, remove the attribute from the source data, and use a tree-based classifier to construct a query that the analyst might issue to recover the attribute. For each data source, we also provide simulated user-defined costs for available curation tasks.

Product Data. We used the product search APIs of two major electronics retailers [1, 5] to extract product data for a total of 586 items (346 and 240 items respectively). The products extracted fall into three categories: TVs, cell phones and laptops. There are ten attributes in the schema of each data source. We randomly replaced 45% of the data values with `NULL`, and coerce both data sets into the schema of a third retailer's search API [3]. On this data-set, we simulate an analyst trying to predict what factors go into a good product rating. We trained a tree based classifier on the partial data, used the resulting decision tree to simulate the analyst's query:

```
SELECT * FROM products
WHERE brand in (4,5,6,7) AND category in (1,2,3)
AND totalReviews < 3 AND instoreAvailability = 0
AND (onsale_clearance = 0 OR (quantityAvailableHint = 0
         AND shippingCost in (0,1,2,3,4)));
```

Curation tasks fall into four categories: Trivial schema matching tasks, simple data gathering of boolean values like item availability, more detailed data gathering of values like strings, and more open-ended data gathering tasks such as soliciting item reviews from focus groups. We assign the cost of these four curation tasks to be 1, 5, 10, and 30 units of effort respectively.

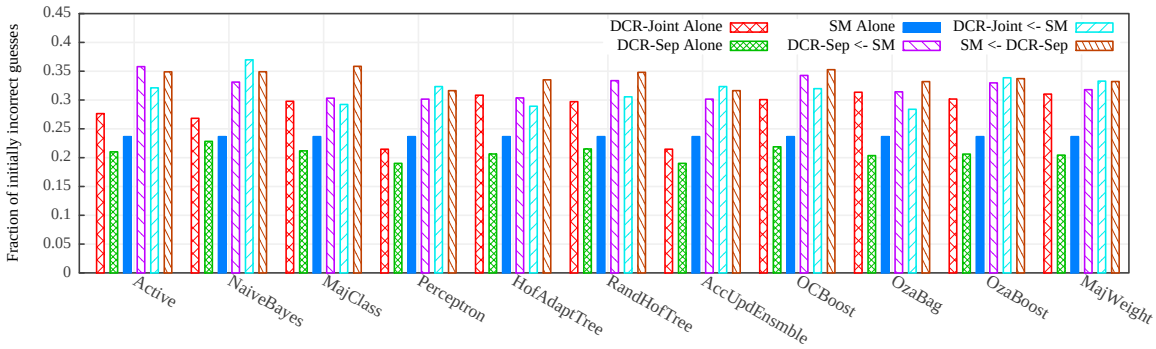


Figure 3-9: Composability of schema matching and domain repair for 11 classifiers (Product Data)

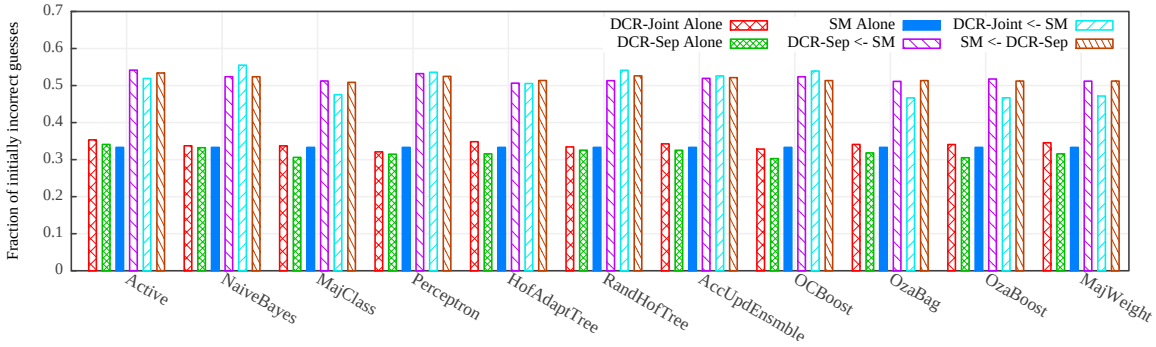


Figure 3-10: Composability of schema matching and domain repair for 11 classifiers (Credit Data)

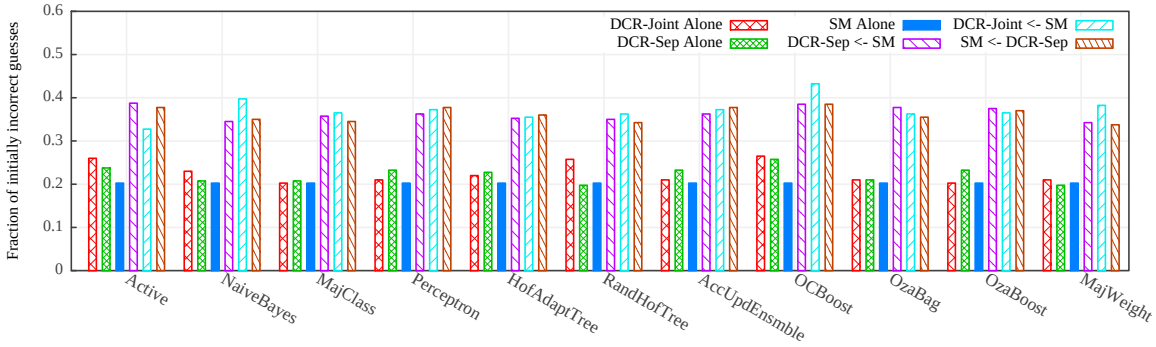


Figure 3-11: Composability of schema matching and domain repair for 11 classifiers (Real Estate Data)

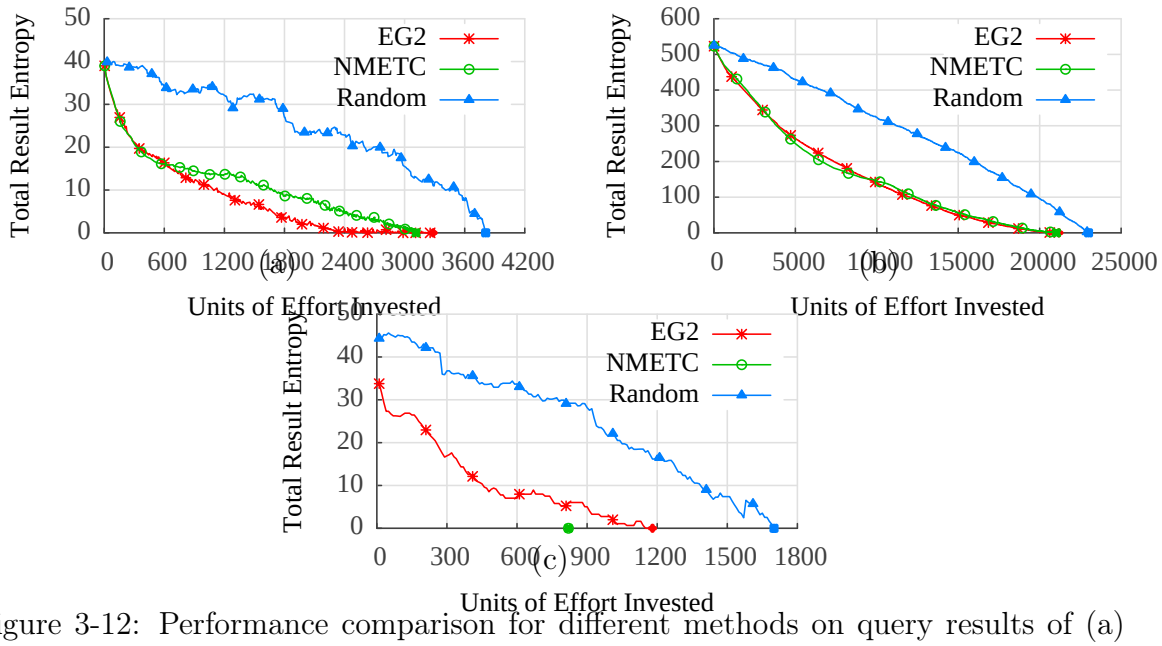


Figure 3-12: Performance comparison for different methods on query results of (a) product, (b) credit, and (c) real estate data-sets. Detailed step-by-step performance for the naive strategy is computationally infeasible for the real estate data-set, so only final results are shown.

Credit Data. We used the German and Japanese Credit Data-sets from the UCI data repository [88]. These data sets contain 1000 and 125 items, respectively, and have 20 and 8 attributes, respectively. As in the product dataset, we randomly replaced 45% of data values with `NULL` values. The German data is coerced into the schema of the Japanese data set. We simulate an analyst searching for low-risk customers by using the following classifier-constructed query:

```
SELECT * FROM PD
WHERE (purchase_item < 0.5 AND monthly_payment >= 3.5
       AND num_of_years_in_company in (2,3) )
   OR (num_of_months >= 6.5 AND married_gender >= 2.5);
```

In addition to trivial schema-matching tasks, there are two kinds of missing attributes: Some attributes can be computed from other values (e.g., a customer's monthly payment can be computed from the total loan value and duration) or re-

trieved from other parts of the bank. Other attributes require personal information about the client. We set the cost of these three classes of task to be 1, 10, and 20 respectively.

Real Estate Data. We obtain house listing information from five real estate websites [4]. Unlike the prior cases, where the number of data-sets is small and the number of records per data set is comparatively large, the Real Estate data set emulates web-tables where the number of data sets is comparatively large and the number of records per data set is small. We further reduce the data size by randomly sampling only 20 items from each dataset. As above, 45% of data values are replaced by `NULL`. All source data is coerced into a globally selected target-schema. For this data set, we simulate an analyst trying to identify houses likely to have a price rating of 3 out of 4 points, where all curation tasks have a flat cost of 1:

```
SELECT * FROM PD WHERE Baths < 2.5
        AND (Beds >= 3.5 OR Garage >= 2.5);
```

Distances	**Correct Matches by Index**								
Metric	1	2	3	4	5	6	...	9	n/a
Levenstein	24	2	2	0	0	1	0	1	0
JaroWinkler	20	4	2	1	1	0	0	1	1
NGram	25	0	3	1	0	0	0	0	1

Figure 3-13: Distance evaluated by the index of the correct match in the ranked list of matches output by the algorithm, or n/a if the correct match was discarded. Results include 30 test cases.

3.5.2 Lens Configuration

For each experiment, we simulate an analyst using the Domain-Constraint Repair and Schema Matching lenses described in Section 3.1.2. We use the values randomly removed from each data-set as ground truths for evaluating the Domain-Constraint Repair lens, and a manually defined mapping as ground truth for evaluating the

Schema Matching lens. Both lenses define curation tasks as summarized in the description of each data-set.

Schema Matching. We employ a combination of schema matchers [100] that hybridize structure-level and element-level matchers. We first use constraint-based (data type and data range) filters to eliminate candidate matches, and then use the Levenstein, JaroWinkler, and NGram distance metrics to rank attribute matches based on string similarity with a threshold. The performance of these three strategies is shown in Figure 3-13. We take an average of the similarity scores from the three distance metrics and normalize them to approximate the probability of a match.

Domain Constraint Repair. We use incremental classifiers from the massive online analysis (MOA) framework [25] for the Domain-Constraint Repair lens. We use classifiers in five categories: active, bayes, stochastic gradient descent, ensemble and tree. For each attribute in the source table, we train a classifier using tuples in which the value is not missing. The estimation results for missing values are probability distributions of all candidate repairs.

3.5.3 Ranking Curation Tasks

We compare three ranking policies over curation tasks (one per variable v_i). Each policy implements a ranking over the available curation tasks, the top-ranked task is performed. Curation costs are as listed in Section 3.5.1.

NMETC. This (naive, exponential-time) policy calculates an optimal long-term strategy based on repeatedly selecting the variable that minimizes the global expected total cost of obtaining a deterministic value. Potential curation tasks are ranked in descending order of their expected total cost, weighted over all possible paths through the decision tree.

Greedy (CPI). CPI based policies rank curation tasks in ascending order of CPI. All four CPI-based metrics produce virtually identical results for each of our test cases, so only results for the EG2 implementation of CPI are shown. The scoring function for the greedy strategy is the CPI itself.

Random. The random strategy ranks curation tasks in a random order, and provides a baseline for other methods.

3.5.4 Lens Composition

We first explore the default (i.e., pre-feedback) behavior of lenses under composition. Of interest to us are three questions: (i) Does the machine learning model used for domain-constraint repair matter? (ii) Can lenses be composed together safely? and (iii) Does the order in which lenses are composed matter? We evaluate the accuracy of the output of Schema Matching (**SM**) and Domain Constraint Repair (**DCR**) lenses applied to each data set. Figures 3-9, 3-10, and 3-11 show the fraction of cells in the output of each query that correspond to ground truth results *before any feedback is gathered*. Our results include two variants of Domain-Constraint Repair, one where all data sources are combined before being repaired (**DCR-Joint**), and one where all data sources are repaired independently (**DCR-Sep**). We consider three different lens combinations: **DCR-Joint** or **DCR-Sep** applied to the output of **SM** (**DCR-Joint** $\leftarrow$ **SM** and **DCR-Sep** $\leftarrow$ **SM**, respectively), and **SM** applied to the output of **DCR-Sep** (**SM** $\leftarrow$ **DCR-Sep**). The remaining combination is not possible, as **DCR-Joint** requires **SM** first to create a unified schema. For comparison, we also present results for each lens alone, using ground truth values for the output of the other lens. Performance results are shown for 11 different machine learning models from the MOA framework.

In general, the performance of different orderings of lenses appears to differ by only a small amount, generally under 5%. An exception appears in the Product data set (Figure 3-9), where we can see for all estimation methods, applying **SM** first and then applying **DCR-Joint** produces the best results. By being trained on both data-sets together, **DCR** is able to detect and correct some schema matching errors. Moreover, in all cases, the combined error of composing both lenses is lower than the error introduced by either lens individually. This shows that composing different lenses is feasible. By comparison, the Credit data set (Figure 3-10) is extremely noisy — both lenses have initial error rates around 34%. Hence, too much noise exists in the data, and different lens orderings have little effect.

The observation above shows that reordering lenses can be beneficial in some cases. Given analyses of lenses, we can help users reorder lenses to achieve better accuracy. Another observation is that when the data is sufficiently correlated for **DCR** to have relatively small error rates, the error rate of **DCR-Joint** is typically lower than **DCR-Sep**. Intuitively, if inter-attribute correlations from different data sets are similar, **DCR-Joint** is effectively being trained on a larger dataset.

3.5.5 On-Demand ETL

We next study the effectiveness of On-Demand ETL and CPI-based heuristics. To study the efficacy of our CPI-based approach, we investigate the performance of different ranking strategies on product,credit, and real estate data-sets. We use the same basic setup as described above for each data-set. We present results using **DCR-Joint** applied to **SM**, but all three composition orders behave similarly.

Figure 3-12 shows the total entropy remaining in the query results after multiple rounds of feedback, in which the analyst repeatedly performs the curation task with

best score. The rightmost dot for each line denotes the point at which the noise in the query result relation reaches zero. Since the analyst may have a limited budget to improve the quality of very noisy query results, the goal is to provide the highest level of noise reduction with as low a total cost as possible, or in other words to create a curve with as little volume under the curve as possible.

EG2 denotes the greedy EG2-based CPI heuristic (all other CPI heuristics behave almost identically). The curve is very steep until the final curation tasks, allowing EG2 to produce high-quality results with minimal investment.

NMETC denotes the naive brute-force cost-optimization strategy, while **Random** denotes a completely random ordering of curation tasks. Although the brute-force strategy produces a completely reliable result at the lowest cost, it does so at the expense of short-term benefits. For the product data-sets, a result with virtually no entropy is reached after 24,000 units of cost, while the brute force strategy requires over 30,000 units. Although NMETC requires the lowest cost to obtain a deterministic query result, it may not be optimal for a limited budget or when the user's value function is not known.

3.5.6 Conclusions

In conclusion, composing lens experiment shows that composing different lenses is feasible by showing that the combined error of composing both lenses is lower than the error introduced by either lens individually. It also shows a potential beneficial in reordering lenses. On-Demand ETL experiment shows without budget limit, our method EG2 achieves a total cost that is sufficiently close to the naive brute-force cost-optimization strategy NMETC. When we have limited budget, EG2 shows a steeper curve than NMETC in most of the cases, allowing EG2 to produce high-

quality results with minimal investment.

Chapter 4

Preliminaries for CIA

For the purpose of illustration, I use Bayesian Networks as a representative probabilistic graphical model. Although my focus here is inference on directed graphical models (i.e. Bayesian networks), the same techniques can be easily adapted for inference in other graphical models .

4.1 Bayesian Networks

Complex systems can often be characterized by multiple interrelated properties. For example, in a medical diagnostics system, a patient might have properties including symptoms, diagnostic test results, and personal habits or predispositions for some diseases. These properties can be expressed for each patient as a set of interrelated random variables. We write sets of random variables in bold (e.g., $\mathbf{X} = \{X_i\}$). Denote by $p(\mathbf{X})$ the probability distribution of $X_i \in \mathbf{X}$ and by $p(x)$ the probability measure of the event $\{X_i = x\}$. Let $\mathbf{X} \backslash \mathbf{Y}$ denote the set of variables that belong to $\mathbf{X}$ but do not belong to $\mathbf{Y}$.

A Bayesian network (BN) represents a joint probability distribution over a set of variables $\mathbf{X}$ as a directed acyclic graph. Each node of the graph represents a random

variable X_i in $\mathbf{X}$. The parents of X_i are denoted by $pa(X_i)$, the children of X_i are denoted by $ch(X_i)$.

A Bayesian network compactly encodes a joint probability distribution using the Markov condition: Given a variable's parents, the variable is independent of all of its non-descendants in the graph. Thus, the full joint distribution is given as:

$$P(\mathbf{X}) = \prod_i P(X_i|pa(X_i))$$

Every random variable X_i is associated with a conditional probability distribution $P(X_i|pa(X_i))$. The joint probability distribution is factorized into a set of $P(X_i|pa(X_i))$ called factors denoted by ϕ_i or factor tables if X_i is discrete. Denote by $scope(\phi_i)$ the variables in a factor ϕ_i. Finally, I use $attrs(\phi_i) = scope(\phi_i) \cup \{p_{\phi_i}\}$ to denote the attributes of the corresponding factor table: the variables in the factor's scope and the probability of a given assignment to X_i given fixed assignments for its parents. A full BN can then be expressed as the 2-tuple $\mathcal{B} = (\mathcal{G}(\mathbf{X}), \Phi)$, consisting of the graph and the set of all factors.

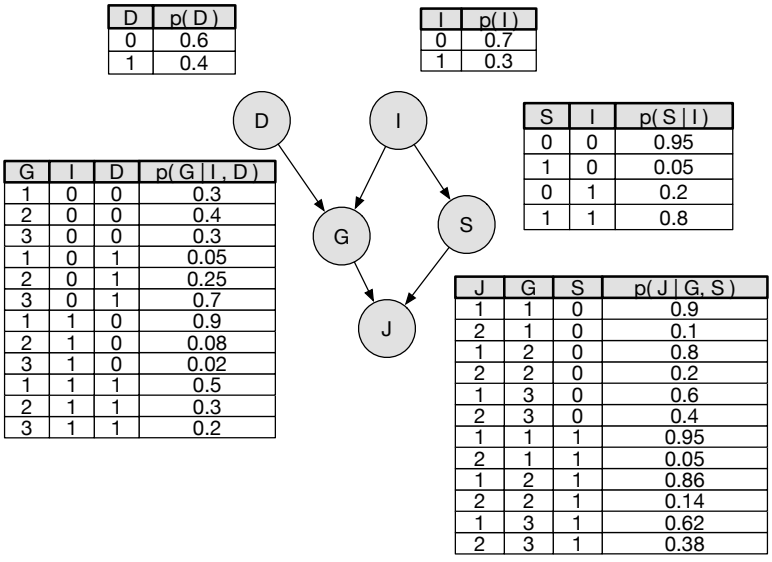


Figure 4-1: A simple Student Bayesian network [82]

Example 11 *Consider five random variables **I**ntelligence, **D**ifficulty, **G**rade, **S**AT, and **J**ob in a Student Bayesian network. The four variables I, D, S, J have two possible values, while G has 3. A relation with* $2 \cdot 2 \cdot 2 \cdot 2 \cdot 3 = 48$ *rows is needed to represent this joint probability distribution. Through the Markov condition, the graph can be factorized into the smaller Bayesian network given in Figure 4-1. For a graph with a large number of variables with large domains, factorization can reduce the size significantly.*

4.2 Inference

Inference in BNs usually involves computing the posterior marginal for a set of query variables $\mathbf{X}_q$ given a set of evidence, denoted by E. For example, $E = \{X_1 = x_1, X_3 = x_3\}$ fixes the values of variables X_1 and X_3. Denote by $\mathbf{X}_E$ the set of observed variables (e.g., $\mathbf{X}_E = \{X_1, X_3\}$). The posterior probability of $\mathbf{X}_q$ given E is

$$P(\mathbf{X}_q|E) = \frac{P(\mathbf{X}_q, E)}{P(E)} = \frac{\sum_{\mathbf{X}\setminus\{\mathbf{X}_q,\mathbf{X}_E\}} P(\mathbf{X})}{\sum_{\mathbf{X}\setminus\mathbf{X}_E} P(\mathbf{X})}.$$

The marginalization of $X_1 \ldots X_i$ over a joint probability distribution is equivalent to a select-join-aggregate query computed over the ancestors of $X_1 \ldots X_i$:

```
SELECT X_1,...,X_i, SUM(p_1 * ... * p_N) AS prob
FROM factor_1 NATURAL JOIN ...  NATURAL JOIN factor_N
WHERE E_1 = e_1 AND ... AND E_k = e_k
GROUP BY X_1,...,X_i;
```

Applying evidence to a graphical model is computationally straightforward and produces a strictly simpler graphical model. As a result, without loss of generality, I ignore evidence and focus exclusively on straightforward inference queries of the form

$P(\mathbf{X}_q)$.

4.2.1 Exact Inference

The exact inference methods introduced in this section all make use of the fundamental insight that the factorization of the distribution allows us to perform local operations on the factors, rather than directly calculate the whole joint distribution.

Variable Elimination. Variable elimination mirrors aggregation push-down [34], a common query optimization technique. The idea is to avoid the exponential blowup in the size of intermediate, joint distribution tables by pushing aggregation down through joins over individual factor tables. As in query optimization, join ordering plays a crucial role in variable elimination, as inference queries often have join graphs with high hypertree width. Intermediate materialized aggregates in VE are typically called separators (denoted S), intermediate (materialized) joins are called cliques (C), variables aggregated away between a clique and the following separator are called clique variables (denoted $var(C)$), and their inputs are called clique factors.

The variable elimination algorithm can be described as follows. First generate an ordering for the N non-observed, non-query variables. Then place all factor tables in a pool of factors. Then for i from i to N do: 1. Create a data strcuture C_i called **clique**, which contains the variable X_i called the **clique variable** and all factor tables that contain the clique variable, called the **clique factors**. 2. Multiply the factors in C_i. The result factor table is called C_i **'s cluster**. I abuse the notation and use C_i to represent both the clique and the clique's cluster. 3. Sum out X_i from C_i 's cluster. The result is called C_i **'s separator** denoted by S_i. 4. Place the clique separator in the factor pool. In the end, collect all the factors that contain the query variables into a clique C_q. Multiply the factors in C_q and normalize the result.

Example 12 *The marginal probability distribution of J in Figure 4-1 can be expressed by* $p(J) = \sum_{D,I,S,G} p(D, I, S, G, J)$. *We choose to first marginalize out D by constructing* C_D*'s separator* S_D*:*

$$S_D[G, I] = \sum_D C_D[D, G, I] = \sum_D \phi_D[D] \bowtie_D \phi_G[D, G, I]$$

Next, we marginalize out I by computing

$$S_I[G, S] = \sum_I C_I[G, I, S] = \sum_I \boldsymbol{S}_D[G, I] \bowtie_I \phi_I[I] \bowtie \phi_S[I, S]$$

The marginalization of G and S follows a similar pattern, leaving us with $C_q = S_S[J] = p(J)$.

Each time we pick a variable X_i to marginalize, we obtain a factor τ_i whose scope is $\texttt{scope}(\psi_i) - \{X_i\}$. The limiting factor in the computational cost of obtaining a separator is enumerating the rows of the clique. Assuming that the distribution over each variable X_i has N possible outcomes ($|dom(X_i)| = N$), the cost of computing a separator S with clique C will be $O(N^{|scope(C)|})$. Tree-width in graphical models (related to query hypertree width) is the size of the largest clique's scope ($\max_C(|scope(C)|)$), making variable elimination exponential-cost in the graph's tree-width.

Belief Propagation. Belief propagation generalizes variable elimination by allowing information to flow in both directions along the graph. Messages are sent along each cluster's separator by summing out all uncommon variables between the two clusters. The process creates, for each variable in the graph, its full conditional probability given all other variables in the graph. Figure 4-2 shows the message passing process for the graph in Figure 4-1. Although belief propagation is more efficient for performing multiple simultaneous inference operations in parallel, for singleton tasks

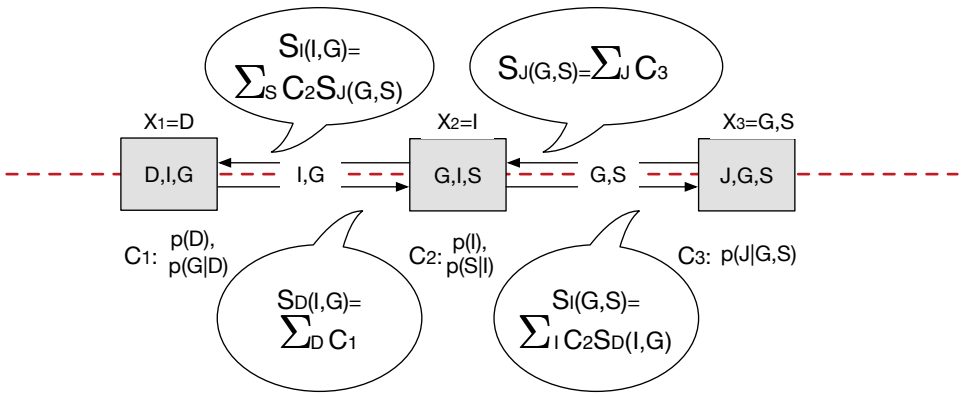


Figure 4-2: Clique tree for Student BN graph in Figure 4-1

(only query one variable) it is a factor of two slower than variable elimination.

4.2.2 Approximate Inference

The inference algorithms introduced in this section approximate the joint distribution as a set of instantiations or particles to variables in the network. The algorithms generate the set of particles to represent the overall probability distribution.

Forward Sampling. Froward sampling samples variables in topological order of the graph. When a variable is being sampled, all of its parents will have already been sampled, reducing the variable's factor table to a single-variable distribution. The quality of the estimate P(Y=y) using forward sampling depends heavily on the number of samples generated. Using *Hoeffding's bound* [65], we can analyze the number of samples required to obtain performance guarantees. We denote the set of samples generated by forward sampling D=X[1],..X[N]. From Hoeffding's bound, we have

$$P_D(P_D(y) \notin [P(y) - \epsilon, P(y) + \epsilon]) \leq 2e^{-2N\epsilon^2} \leq \delta.$$

This analysis gives an estimate of how many samples are required to achieve an estimate $P_D(y)$ whose error is bounded by ϵ, with probability at least 1-δ. Therefore,

the required size to get an estimation $P_D(y)$ with (ϵ,δ) bound is N $\geq \frac{ln(2/\delta)}{2\epsilon^2}$. If the true distribution P(y) is very small, it is naturally to think we will need more samples to guarantee that the estimation is close to the true probability. To take consideration of P(y) in the analysis of ϵ,δ bound, we can apply *Chernoff bound* [35] to conclude that $P_D(y)$ is with a relative error ϵ of the true value P(y).

$$P_D(P_D(y) \notin P(y)(1 \pm \epsilon)) \leq 2e^{-NP(y)\epsilon^2/3} \leq \delta.$$

Thus, the number of samples needed is N $\geq 3\frac{ln(2/\delta)}{P(y)\epsilon^2}$ which grows inversely with the probability of P(y). Forward sampling works well for estimating high probability groups, but for conditional probabilities $P(y|E = e)$, the sampling process becomes much harder, requiring the use of rejection sampling to discard forward samples that conflict with the condition.

Likelihood weighting. method improves forward sampling by generating samples more relevant to evidence. Instead of considering each sample equally weighed in forward sampling, likelihood weighting weights each sample by the likelihood of the evidence accumulated throughout the sampling process. The limitation of this method is that since it still follows topological order of sampling, an evidence node affects the sampling only for nodes that are its descendants. In cases where the evidence is at the leaves of the network, it is the same as sampling from the prior distribution which is far from the desired posterior.

Markov Chain Monte Carlo Inference. MCMC is a family of sampling techniques that generate sequences of samples. Intuitively, the first element in the sequence is drawn from the prior and successive samples are drawn from distributions that get increasingly closer to the posterior. For example, we might draw one assignment of values in $\mathbf{X}$ with each variable X_i following the conditional probability

distribution $p(X_i|pa(X_i))$ in the topological order of the graph. Then, we iteratively re-sample one variable's value at a time according to its factor table, given the current assignments for its parents and children. The longer we continue re-sampling, the less the sample is biased by its initial value. We use **Gibbs sampling** as a representative MCMC inference algorithm.

Loopy Belief Propagation. Loopy belief propagation is the same as belief propagation, but operates on a loopy cluster graph instead of a clique tree. This change makes the cluster smaller than those in clique tree and makes message passing steps less expensive. There is a trade-off between cost and accuracy in loopy-belief propagation, as join graphs that allow fast propagation may create a poor approximation of the result.

4.3 Online Aggregation

Starting with work by Hellerstein et.al. [63], Olken [94], and others, a large body of literature has been developed for so-called online aggregation (OLA) or approximate query processing (AQP) systems. Such systems replace pipeline-blocking operators like join and aggregate with sampling-based operators that permit approximate or partial results to be produced immediately and iteratively refined the longer the user is willing to wait. The work most closely related to our own efforts is on OLA [61–63]. OLA systems use query evaluation strategies that estimate and iteratively refines the output of aggregate-joins. Given enough time, in most systems, the evaluation strategy eventually converges to a correct result. As in random sampling, $\epsilon - \delta$ bounds can be obtained, for example using Hoeffding's inequality.

A key challenge arising in OLA is how to efficiently generate samples of source data. Sampling without replacement allows the algorithm to converge to the correct

result once all samples have been exhausted, but has high space requirements, as it is necessary to keep track of the sampling order. Conversely, sampling with replacement is not guaranteed to ever converge to the correct answer. One of our key contributions in this paper is a specialization of OLA to graphical models called Cyclic Sampling, which permits sampling without replacement using only constant-space. Numerous other systems have since adapted and improved on the idea of OLA. Aqua [6] uses key constraints for improved stratified sampling. BlinkDB [7, 8] and Derby [79] maintain pre-materialized stratified samples to rapidly answer approximate queries. GLADE [104] and DBO [44, 72] exploit file buffering to opportunistically generate samples of a query result in the course of normal query evaluation.

Chapter 5

Interactive Data Analysis For Probabilistic Graphical Models - CIA

5.1 Introduction

As discussed in Chapter 1, the goal of CIA is, given a fixed time bound, to produce a bounded approximate inference result, but will also terminate early if it is possible to converge to an exact result. The challenges for such a goal are: (1) to guarantee the time complexity to obtain exact result is competitive with existing classic exact inference algorithms, (2) to guarantee the approximation accuracy is competitive with common approximate techniques. To solve this challenge, I propose two specific CIAs that use the relationship between inference and select-join-aggregate queries to build on database techniques for OLA [63]. My algorithms specialize OLA to two unique requirements of graphical inference: dense data and wide joins. In classical group-by aggregate queries, the joint domain of the group-by attributes is sparse: Tables in a typical database only have a small portion of their active domain populated. Furthermore, classical database engines are optimized for queries involving a small number of

large input tables. Conversely, in graphical inference, each "table" is small and dense and there are usually a large number of tables with a much more complicated join graph.

The density of the input tables (and by extension, all intermediate relations) makes it practical to sample directly from the output of a join, since each sample from the active domain of the output relation is likely to hit a row that is present and non-zero. Hence, our first, naive CIA samples directly from the output of the select-join component of the inference query, using the resulting samples to predict the aggregate query result. To ensure convergence, I leverage a class of pseudorandom number generators called Linear Congruential Generators [98, 105] (LCGs). The cyclicity of LCGs has been previously used for coordinating distributed simulations [18]. Here, I use them to perform random sampling from join outputs *without* replacement, allowing us to efficiently iterate through the join outputs in a shuffled order. These samples produce bounded-error estimates of aggregate values. After the LCG completes one full cycle, every row of the join has been sampled exactly once and the result is exact.

Unfortunately, the domain of the join output for an inference query can be quite large and this naive approach converges slowly. To improve convergence rates, I propose a new online join algorithm called *Leaky Joins* that produces samples of a query's result in the course of normally evaluating the query. Systems for relational OLA (e.g., [44, 63, 72]) frequently assume that memory is the bottleneck. Instead, Leaky Joins are optimized for small input tables that make inference more frequently compute-bound than IO-bound. Furthermore, the density of the input (and intermediate) tables makes it possible to use predictable, deterministic addressing schemes. As a result, Leaky Joins can obtain unbiased samples efficiently without needing to assume a lack of correlation between attributes in the input.

The Leaky Joins algorithm starts with a classical bushy query plan. Joins are

evaluated in parallel, essentially *"leaking"* estimates for intermediate aggregated values — marginal probabilities in our motivating use case — from one intermediate table to the next. One full cycle through a LCG is guaranteed to produce an exact result for joins with exact inputs available. Thus, initially only the intermediate tables closest to the leaves can produce exact results. As sampling on these tables completes a full cycle, they are marked as stable, sampling on them stops, and the tier above them is permitted to converge. In addition to guaranteeing convergence of the final result, we are also able to provide confidence bounds on the approximate results prior to convergence. As we show in our experiments, the algorithm satisfies desiderata for a useful convergent-inference algorithm: *computation performance* competitive with exact inference on simple graphs, and *progressive accuracy* competitive with approximate inference on complex graphs.

Our main motivation is to generate a new type of inference algorithm for graphical inference in databases. Nevertheless, we observe that Leaky Joins can be adapted to any aggregate queries over small but dense tables.

5.2 Convergent Inference

Running-time for variable elimination $O(N^{|scope(C)|})$, is dominated by tree-width, and strongly depends on the elimination ordering (already an NP-hard problem [82]). Since the running time grows exponentially in the size of largest clique cluster C_{max}, the running complexity can have high variance depending on the order. Because the cost is exponential, even a small increase in complexity can change the runtime of variable elimination from seconds to hours, or even days. In short, predicting whether an exact solution is feasible is hard enough that most systems simply rely exclusively on approximation algorithms. On other hand, approximate inference may

get asymptotically close to an answer, but it will never fully converge. Thus, most applications that benefit from exact results must rely on either human intuition to decide.

The goal and first contribution of this paper is to introduce *convergent inference*, a specialized form of approximate inference algorithm that is guaranteed to eventually converge to an exact result. In this section, we develop the idea of convergent inference and propose several convergent inference algorithms, or CIAs. A CIA eventually produces an exact result, but can be interrupted at any time to quickly produce a bounded approximation. More precisely, a CIA should satisfy the following conditions: (1) After a fixed period of time t, a CIA can provide approximate results with $\epsilon - \delta$ error bounds, such that $P\left(|P_t - P_{exact}| < \epsilon\right) > 1 - \delta$; and (2) A CIA can will obtain the exact result P_{exact} in a bounded time t_{exact}.

Ideally, we would also like a CIA to satisfy two additional conditions: (3) The time complexity required by a good CIA to obtain an exact result should be competitive with variable elimination; and (4) The quality of the approximation produced by a good CIA should be competitive with the approximation produced by a strictly approximate inference algorithm given the same amount of time.

We first introduce a fundamental algorithm called cyclic sampling which performs pseudo-random sampling without replacement from the joint probability distribution of the graph. This algorithm is guaranteed to converge, but requires an exponential number of samples to do so. We then present an improved CIA based on classical aggregate-join query processing that relies on a novel "leaky join" operator. Finally, we discuss lessons learned in a failed attempt to combine cyclic sampling with state-of-the-art techniques for incremental view maintenance.

All three of our approaches draw on the relationship between graphical inference and aggregate query processing. However, though the problems are similar, we re-

emphasize that there are several ways in which graphical inference queries violate assumptions made in classical database query-processing settings. First, conditional probability distributions are frequently dense, resulting in many-many relationships on join attributes. Correlations between variables are also common, so graphical models often have high tree widths. By comparison, the common case for join and aggregation queries is join graphs with a far smaller number of tables, simpler (e.g., foreign key) predicates, and typically low tree widths.

Finally, we note that although we use graphical models as a driving application, similar violations occur in other database applications (e.g., scientific databases [113]). The algorithms we present could be adapted for use in these settings as well.

5.2.1 Cyclic Sampling

I first discuss a naive form of convergent inference called cyclic sampling that forms the basis for each of our approaches. Recall that each intermediate table (the separator) is an aggregate computed over a join of factor tables (the clique), and that the domain of the clique is (or is very nearly) a cartesian product of the attributes in its scope. In principle, one could compute an entire separator table by scanning over the rows of its clique.

We note that input factor tables and the separator tables are typically small enough to remain in memory. Thus, using array-indexed storage for the clique tables is feasible and the cost of accessing one row of the clique is a constant. Consequently, efficient random sampling on the joint probability distribution is possible, and the marginal probabilities of interest can be incrementally approximated as in OLA [63].

The key insight of cyclic sampling is that if this random sampling is performed without replacement, it will eventually converge to an exact result if we reach a point

where each row of the clique has been sampled exactly once. Unfortunately, sampling without replacement typically has space complexity linear in the number of items to be sampled, which is exponential in the number of variables. Fortunately for graphical inference, a there exists a class of so-called cyclic pseudorandom number generators that iteratively construct pseudorandom sequences of non-repeating integers in constant space.

A cyclic pseudorandom number generator generates a sequence of non-repeating numbers in the range $[0, m)$ for some given period m with members that exhibit minimal pairwise correlation. We use Linear Congruential Generators (LCGs) [98, 105], which generate a sequence of semi-random numbers with a discontinuous piecewise linear equation defined by the recurrence relation:

$$X_n = (aX_{n-1} + b) \mod m \tag{5.1}$$

Here X_n is the nth number of the sequence, and X_{n-1} is the previous number of the sequence. The variables a, b and m are constants: a is called the multiplier, b the increment, and m the modulus. The key, or seed, is the value of X_0, selected uniformly at random between 0 and m. In general, a LCG has a period no greater than m. However, if a, b, and m are properly chosen, then the generator will have a period of exactly m. This is referred to as a maximal period generator, and there have been several approaches to choosing constants for maximal period generators [80, 86]. In our system, we follow the Hull-Dobell Theorem [110], which states that an LCG will be maximal if

1. m and the offset b are relatively prime.

2. $a - 1$ is divisible by all prime factors of m.

3. $a - 1$ is divisible by 4 if m is divisible by 4.

LCGs are fast and require only constant memory. With a proper choice of parameters a, b and m, a LCG can produce maximal period generators and pass formal tests for randomness. Parameter selection is outlined in Algorithm 5.

Algorithm 5 `InitLCG(totalSamples)`

Require: *totalSamples*: the total number of samples
Ensure: a, b, m: LCG Parameters
1: $m \leftarrow totalSamples$
2: $S \leftarrow$ prime factors of m
3: **for all** s **in** S **do**
4: $\quad a \leftarrow a \times s$
5: **if** $m = 0 \mod 4$ **and** $a \neq 0 \mod 4$ **then**
6: $\quad a \leftarrow 4a + 1$
7: $b \leftarrow$ any coprime of m smaller than m

The cyclic sampling process itself is shown in Algorithm 6. Given a Bayesian network $\mathcal{B}$=($\mathcal{G}(\mathbf{X})$,Φ), and a marginal probability query Q = $p(\mathbf{X}_q)$, we first construct the LCG sampling parameters (lines 1-5): a, b and m according to Hull-Dobell Theorem for the total number of samples in the joint probability distribution p($\mathbf{X}$). The sampling process (starts at Line 16) constructs an index by obtaining the next value from the LCG (line 7) and decomposes the index into an assignment for each variable (line 10). We calculate the joint probability of p($\mathbf{X}$) for this assignment (line 13) add this probability to the corresponding result row, and increment the sample count.

At any point, the algorithm may be interrupted and each individual probability in $p(\mathbf{X}_q)$ may be estimated from the accumulated probability mass $p(x)$:

$$p(\mathbf{X}_q) = \frac{\prod_{X_j \in \mathbf{X}} |dom(X_j)|}{count_x} \cdot p(x)$$

Cyclic sampling promises to be a good foundation for CIA, but must satisfy the two constraints. First, it needs to provide an epsilon-delta approximation in a fixed

period of time. In classical OLA and some approximate inference algorithms, samples generated are independently and identically distributed. As a result, Hoeffding's inequality can provide accuracy guarantees. In CIAs, samples are generated without replacement. We need to provide an $\epsilon - \delta$ approximation under this assumption. Second, cyclic sampling needs to eventually converge to an exact inference result. This requires that we sample all the items in the joint probability distribution exactly once and the samples should be sampled randomly.

Algorithm 6 `CyclicSampling`($\mathcal{B}$, Q)

Require: A bayes net $\mathcal{B}$=($\mathcal{G}(\mathbf{X})$,Φ)
Require: A conditional probability query: Q=$P(\mathbf{X_q})$
Ensure: Probabilities for each $\vec{q}$: $\{p_{\vec{q}} = P(\mathbf{X_q}) = \vec{x}_q)\}$
Ensure: The number of samples for each $\vec{q}$: $\{count_{\vec{q}}\}$
1: $totalSamples \leftarrow 1$
2: **for all** ϕ_i **in** Φ **do**
3: $\quad totalSamples = totalSamples * |dom(X_i)|$
4: $index_1 \leftarrow$ `rand_int()` mod $totalSamples$
5: $a, b, m \leftarrow$ `InitLCG`($totalSamples$)
6: **for all** $k \in 0 \ldots totalSamples$ **do**
7: $\quad assignment \leftarrow index_k$
8: $\quad$ `/* De-multiplex the variable assignment */`
9: $\quad$ **for all** $j \in 0 \ldots n$ **do**
10: $\quad\quad x_j \leftarrow assignment \mod |dom(X_j)|$
11: $\quad\quad assignment \leftarrow assignment \div |dom(X_j)|$
12: $\quad$ `/* probability is a product of the factors */`
13: $\quad prob \leftarrow \prod_i \phi_i \left(\pi_{scope(\phi_i)}(\langle\, x_1, \ldots, x_n \,\rangle)\right)$
14: $\quad$ `/* assemble return values */`
15: $\quad \vec{q} \leftarrow \pi_{\mathbf{X_q}}(\, \vec{x}\,)$; $p_{\vec{q}} \leftarrow p_{\vec{q}} + prob$; $count_{\vec{q}} \leftarrow count_{\vec{q}} + 1$
16: $\quad$ `/* step the LCG */`
17: $\quad index_{k+1} \leftarrow (a * index_k + b) \mod m$

Computation Cost. Let n be the number of random variables in $\mathcal{B}$, as a simplification assume w.l.o.g. that each random variable has domain size dom. Calculating the parameters for LCG takes constant time. The sampling process takes $O(|N|)$ time, where $|N|$ is the total number of samples in the reduced joint probability distribution $P(\mathbf{X})$. N can be as large as dom^n, that is exponential in the size of the graph.

Confidence Bound. Classical approximate inference and OLA algorithms use random sampling with replacement, making it possible to use well known accuracy bounds. For example one such bound, based on Hoeffding's inequality [65] establishes a tradeoff between the number of samples needed n, a probabilistic upper bound on the absolute error ϵ, and an upper bound on probably that the bound will be violated δ. Given two values, we can obtain the third.

Hoeffding's inequality for processes that sample with replacement was extended by Serfling et al. [109] for sampling without replacement. Denote by N the total number of samples, P(x) is the true probability distribution after seeing all the samples N. Denote by $P_n(x)$ the approximation of P(x) after n samples. From [109], and given that probabilities are in general bounded as $0 \leq p \leq 1$, we have that:

$$P_n\left(P_n(x) \notin [P(x) - \epsilon, P(x) + \epsilon]\right) \leq \delta \equiv exp\left[\frac{-2n\epsilon^2}{1 - (\frac{n-1}{N-1})}\right] \tag{5.2}$$

In other words, after n samples, there is a $(1 - \delta)$ chance that our estimate $P_n(x)$ is within an error bound ϵ.

5.2.2 Leaky Joins

In Cyclic Sampling, samples are drawn from an extremely large joint relation and require exponential time for convergence. To address this limitation, we first return to Variable Elimination as described in Section 4.2.1. Recall the clique tree representation in Figure 4-2 for the BN in Figure 4-1, where the marginal for the goal variable (J) is produced by clique cluster C_3. Each clique focuses on a single clique variable X_i, and the clique cluster is a product of the separator table to the clique's left and all remaining factor tables containing X_i. As a result, each factor ϕ in $\mathcal{B}$=($\mathcal{G}(\mathbf{X})$,Φ) belongs to exactly one clique cluster. Variable Elimination (the process below the red

line) mirrors classical blocking aggregate-join evaluation, computing each separator table (aggregate) fully and passing it to the right.

The key idea is to create a clique tree as in Variable Elimination, but to allow samples to gradually "leak" through the clique tree rather than computing each separator table as a blocking operation. To accomplish this, we propose a new *Leaky Join* relational operator. A single Leaky Join computes a group-by aggregate over one or more Natural Joins, "online" using cyclic sampling as described above. As the operator is given more cpu-time, its estimate improves. Crucially, Leaky Joins are composable. During evaluation, all Leaky Joins in a query plan are updated in parallel. Thus the quality of a Leaky Join operator's estimate is based not only on how many samples it has produced, but also on the quality of the estimates of its input tables.

Algorithm 7 gives an evaluation strategy for inference queries using Leaky Joins. Abstractly, an evaluation plan consists of a set of intermediate tables for each clique $C_i \in \mathbf{C}$ (i.e., each intermediate join), and for each separator $S_i \in \mathbf{S}$ (i.e., each intermediate aggregate). Queries are evaluated volcano-style, iterating over the rows of each clique and summing over the product of probabilities as described in Section 4.2. As in Cyclic Sampling, the iteration order is randomized by a LCG (lines 9-13). For each clique C_i, the algorithm samples a row $\vec{x}$ (lines 9-12), computes the marginal probability for that row (line 14), and adds it to its running aggregate for the group $\vec{q}$ that $\vec{x}$ belongs to (line 20). It is necessary to avoid double-counting samples in the second and subsequent cycles of the LCG. Consequently, the algorithm updates the separator using the difference δ_{prob} between the newly computed marginal and the previous version (lines 16-19).

In order to determine progress towards convergence, the algorithm also tracks a sample count for each row of the clique and separator tables (lines 15, 17), as well as

Algorithm 7 `EvaluateLeakyJoins`($\mathcal{B}$, Q)

Require: A bayes net $\mathcal{B} = (\mathcal{G}(\mathbf{X}), \Phi)$
Require: An inference query $Q = P(\mathbf{X_q})$
Ensure: The result separator $S_{target} = P(\mathbf{X_q})$
1: $\langle\ \mathbf{S}, \mathbf{C}\ \rangle \leftarrow$ `assemblePlan`$(\mathcal{B}, \mathbf{X}_q)$
2: **for all** $i \in 1 \ldots |\mathbf{S}|$ **do**
3: $samples_i \leftarrow 0$; $maxSamples_i = |dom(desc(S_i))|$
4: $a_i, b_i, m_i \leftarrow$ `initLCG`$(|dom(C_i)|)$
5: $index_i \leftarrow$ `rand_int`() mod m_i
6: Fill S_i and C_i with $\langle\ prob : 0.0\ ,\ count : 0\ \rangle$
7: **while** there is an i with $samples_i < maxSamples_i$ **do**
8: **for all** i where $samples_i < maxSamples_i$ **do**
9: `/* Step the LCG */`
10: $index_i \leftarrow (a_i * index_i + b_i)$ mod $|dom(\psi_i)|$
11: `/* Demux index as Alg. 6 lines 9-11 */`
12: Get $\vec{x}$ from $index_i$
13: `/* Get the joint probability as Alg. 6 line 13 and get the joint sample count similarly */`
14: $prob \leftarrow \prod_{\phi \in factors(C_i)} \left(\phi\left[\pi_{scope(\phi)}(\vec{X})\right].prob\right)$
15: $count \leftarrow \prod_{\phi \in factors(C_i)} \left(\phi\left[\pi_{scope(\phi)}(\vec{X})\right].count\right)$
16: `/* Compute update deltas */`
17: $\langle\ \delta_{prob}, \delta_{count}\ \rangle = \langle\ prob, count\ \rangle - C_i[\vec{x}]$
18: `/* Apply update deltas */`
19: $C_i[\vec{x}] = C_i[\vec{x}] + \langle\ \delta_{prob}, \delta_{count}\ \rangle$
20: $\vec{q} = \pi_{scope(S_i)}(\vec{x})$; $S_i[\vec{q}] = S_i[\vec{q}] + \langle\ \delta_{prob}, \delta_{count}\ \rangle$
21: $samples_i = samples_i + \delta_{count}$

the total, aggregate count for each separator table (line 21). Informally, this count is the number of distinct tuple lineages represented in the current estimate of the probability value. For a given clique table C_i the maximum value of this count is the product of the sizes of the domains of all variables eliminated (aggregated away) in C_i (line 3). For example, in Figure 4-2, no variables have been eliminated in C_1 so each row of C_1 contains at most one sample. By C_2, the variable D has been eliminated, so each row of C_2 can represent up to $|dom(D)| = 2$ samples. Similarly each row of C_3 represents up to $|dom(D)| \cdot |dom(I)| = 4$ samples. The algorithm uses the sample count as a termination condition. Once a table is fully populated, sampling on it

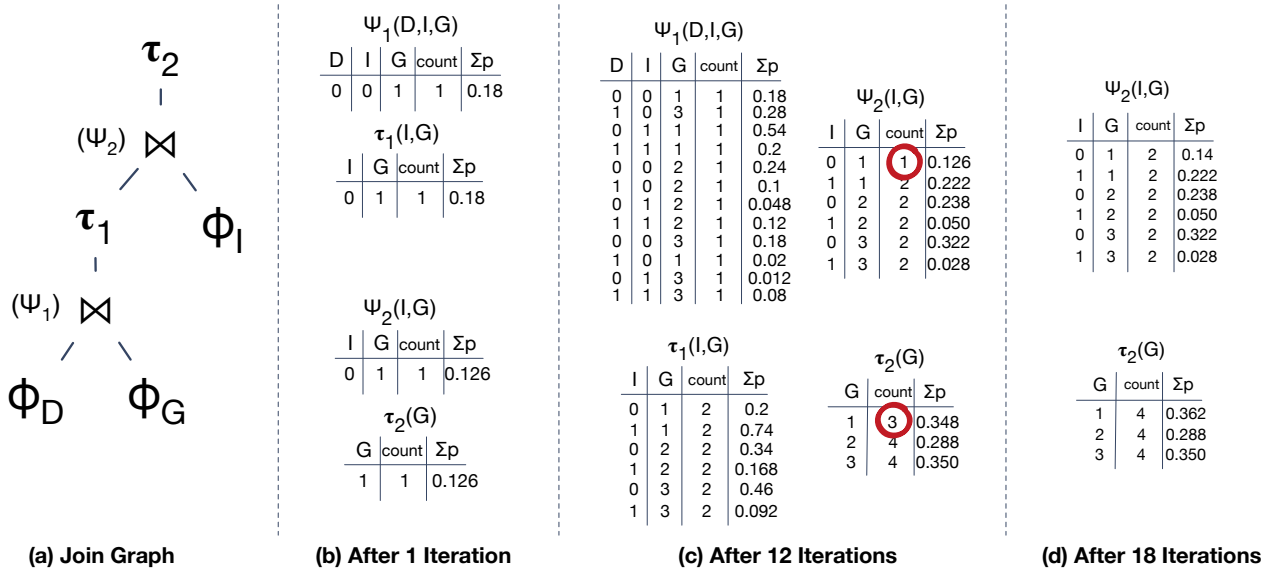


Figure 5-1: Leaky Joins example join graph (a) and the algorithm's state after 1, 12, and 18 iterations (b-d). In the 12-iteration column (c), incomplete sample counts are circled.

stops (line 8). Once all tables are fully populated, the algorithm has converged (line 7).

In short, Leaky Joins work by trickling samples down through each level of the join graph. The cyclic sampler provides flow control and acts as a source of randomization, allowing all stages to produce progressively better estimates in parallel. With each cycle through the samples, improved estimates from the join operator's input are propagated to the next tier of the query.

Example 13 *As an example of Leaky Joins, consider a subset of the graph in Figure 4-1 with only nodes D, I, G and an inference query for $p(G)$. Using classical heuristics from variable elimination, the Leaky Joins algorithm elects to eliminate D first and then I, and as a result assembles the intermediate clique cluster $\boldsymbol{C}$ and clique separator $\boldsymbol{S}$ tables as shown in Figure 5-1. Samples are generated for each intermediate clique cluster one at a time, following the join order: First from $C_1(D, I, G)$ and then $C_2(I, G)$. As shown in Figure 5-1b, the first sample we obtain from C_1 is*

$\langle\ 0, 0, 1\ \rangle$

$$\phi_D(D=0) \cdot \phi_G(D=0, I=0, G=1) = 0.6 \cdot 0.3 = 0.18$$

$S_1(I, G)$ is correspondingly updated with the tuple $\langle\ 0, 1\ \rangle$ with aggregates $\langle\ 1, 0.18\ \rangle$. Then the second sample is drawn from $C_2(I, G)$. For this example, we will assume the random sampler selects $\langle\ 0, 1\ \rangle$, which has probability:

$$S_1(I=0, G=1) \cdot \phi_I(I=0)$$

Although we do not have a precise value for S_1 we can still approximate it at this time and update S_2 accordingly. After 12 samples, C_1 has completed a round of sampling and is ready to be finalized. The state at this point is shown in Figure 5-1c. Note that the approximation of S_2 is still incorrect — The approximation made in step 1 and several following steps resulted in only partial data for $\psi_2(I=1, G=1)$ (circled counts in Fig. 5-1c). However, this error will only persist until the next sample is drawn for $C_2(0, 1)$, at which point the system will have converged to a final result.

Cost Model. We next evaluate the cost of reaching an exact solution using the Leaky Join algorithm. Assume we have k random variables $X_1, ..., X_k$, and the corresponding k factors $\phi_{X_1}, ..., \phi_{X_k}$. Furthermore, assume the variables are already arranged in the optimal elimination order, and we wish to marginalize out variables $X_1, \ldots, X_j$. Leaky joins generates exactly the same set of j cliques and separators as Variable Elimination. Like variable elimination, we can measure the computation complexity by counting multiplication and addition steps. The primary difference between Variable Elimination and Leaky Joins is that some aggregation steps will base on approximations and must be repeated multiple times. Let V denote the cost of constructing the largest joint factor in Variable Elimination (i.e., the time complexity

of Variable Elimination is $O(V)$). After V iterations, the lowest level of the join tree is guaranteed to be finalized. After a successive V iterations, the second level of the tree is guaranteed to be finalized, and so forth. The maximum depth of the join tree is the number of variables k, so a loose bound for the complexity Leaky Joins is $O(kV)$.

Confidence Bound. In Section 5.2.1, we showed an ϵ-δ bound for random sampling without replacement (Formula (5.2)). Here, we extend this result to give a loose bound for Leaky Joins. The primary challenge is that, in addition to sampling errors in the Leaky Join itself, the output can be affected by cumulative error from the join's inputs. We consider the problem recursively. The base case is a clique that reads from only input factors — the lowest level of joins in the query plan. Precise values for inputs are available immediately and Formula (5.2) can be used as-is.

Next, consider a clique C_2 computed from only a single leaky join output. Thus, we can say that $C_2 = S_1 \bowtie \phi$, where ϕ is the natural join of all input factors used by C_2. There are $|dom(S_1)|$ rows in S_1, so after n sampling steps, each row of S_1 will have received $\frac{n}{|dom(S_1)|}$ samples. Denote the maximum number of samples per row of S_1 by $N_1 = \frac{|dom(S_1)|}{|dom(C_1)|}$. Then, by (5.2), all rows in S_1 will have error less than ϵ with probability:

$$\delta_1 \equiv \delta^{|dom(S_1)|} = exp\left[\frac{-2n\epsilon^2 \cdot (N_1 - 1)}{N_1 - \frac{n}{|dom(S_1)|}}\right] \tag{5.3}$$

Let us consider a trivial example where the cumulative error in each row of S_1 is bounded by ϵ:

S_1	X_1	p
	1	$p_1 \pm \epsilon$
	2	$p_2 \pm \epsilon$

ϕ	X_1	p
	1	p_3
	2	p_4

Here, the correct joint probabilities for rows of C_2 are $p_1 \cdot p_3$ and $p_2 \cdot p_4$ respectively. Thus a fully-sampled S_2 (projecting away X_1) will be approximated as $(p_1 \pm \epsilon)p_3 +$

$(p_2 \pm \epsilon)p_4$. The cumulative error in this result is $(p_3+p_4)\epsilon$, or using a pessimistic upper bound of 1 for each p_i, at worst 2ϵ. Generalizing, if one row of S_2 is computed from k rows of C_1, the cumulative error in a given row of S_2 is at most $k\epsilon$. Repeating (5.3), sampling error on S_2 after n rounds will be bounded by ϵ with probability $\delta^{|dom(S_2)|}$. After n rounds of sampling, each row of S_2 will have received $\frac{n}{|dom(S_2)|}$ rows of C_2, so the cumulative error on one row is $\frac{n}{|dom(S_2)|}\epsilon$. Combining (5.2) and (5.3), we get that for one row of S_2:

$$P\left[|p - \mathbb{E}_{n,x}| < \left(1 + \frac{n}{|dom(S_2)|}\right)\epsilon\right] \leq exp\left[\frac{-2\frac{n}{|dom(S_2)|}\epsilon^2 \cdot (N_2 - 1)}{N_2 - \frac{n}{|dom(S_2)|}}\right] \cdot \delta_1$$

The joint probability across all rows of S_2 is thus:

$$exp\left[\frac{-2n\epsilon^2(N_2 - 1)}{N_2 - \frac{n}{|dom(S_2)|}}\right] \cdot \delta_1^{|dom(S_2)|} \equiv \delta_2 \cdot \delta_1^{|dom(S_2)|}$$

Generalizing to any **left-deep** plan, an error ϵ' defined as:

$$\epsilon' = \epsilon \cdot \prod_{i=2}^{|\mathbf{X}|}\left(1 + \frac{n}{|dom(S_i)|}\right) \tag{5.4}$$

is an upper bound on the marginal in $S_{|\mathbf{X}|}$ with probability:

$$\prod_{i=1}^{|\mathbf{X}|}\delta_i^{\left(\prod_{j=1}^{i-1}|dom(S_j)|\right)} = \prod_{i=1}^{|\mathbf{X}|} e^{\left(\left(\prod_{j=1}^{i-1}|dom(S_j)|\right)\frac{-2n\epsilon^2(N_j-1)}{N_j - \frac{n}{|dom(S_j)|}}\right)} \tag{5.5}$$

Consider a slightly more complicated toy example clique $C_4 = S_3 \bowtie S_1$, where both S_3 and S_1 both have bounded error ϵ_1 and ϵ_3 respectively.

C_1	X_1	p
	1	$p_1 \pm \epsilon_1$
	2	$p_2 \pm \epsilon_1$

C_3	X_1	p
	1	$p_3 \pm \epsilon_3$
	2	$p_4 \pm \epsilon_3$

As before, the correct joint probability is $p_1 \cdot p_3 + p_2 \cdot p_4$. Given an $\epsilon' = \max(\epsilon_1, \epsilon_3)$, the estimated probability will be $p_1 \cdot p_3 + p_2 \cdot p_4 + \left(\sum_{i=1}^{4} p_i \epsilon'\right) + \epsilon'^2$. As before, using an upper bound of 1 for each $p_1 \ldots p_4$ bounds the error by:

$$(|dom(S_1)| \cdot |dom(S_3)|)\epsilon' + \epsilon'^2$$

More generally, the predicted value across m source tables is a sum of terms of the form $(p + \epsilon')$, and the overall cumulative error per element of C_1 is bounded as:

$$\epsilon_{cum} = \sum_{i=1}^{m-1} (m \; \mathcal{C} \; i) \epsilon^{m-i}$$

where $m \; \mathcal{C} \; i$ is the combinatorial operator m choose i. Thus re-using Equation (**??**), we can solve the recursive case of a separator with m leaky join inputs that each have error bounded by ϵ' with probability $\delta_{cum} = \prod_i^m \delta_i$. Then the total error on one row of the separator can be bounded by $\epsilon + \epsilon_{cum}$ with probability:

$$exp\left[\frac{-2\frac{n}{|dom(S_2)|}\epsilon^2 \cdot (N_2 - 1)}{N_2 - \frac{n}{|dom(S_2)|}}\right] \cdot \delta \tag{5.6}$$

The joint error is computed exactly as before. To estimate the error on the final result, we apply this formula recursively on the full join plan.

5.3 Lessons Learned From IVM

Materialized views are the precomputed results of a so-called view query. As the inputs to this query are updated, the materialized results are updated in kind. Incremental view maintenance (IVM) techniques identify opportunities to compute these updates more efficiently than re-evaluating the query from scratch. Incremental view

maintenance has already seen some use in Monte Carlo Markov Chain inference [124], and recent advances — so called recursive IVM techniques [10, 81] have made it even more efficient.

Our initial attempts at convergent inference were based on IVM and recursive IVM in particular. It eventually became clear that there was a fundamental disconnect between these techniques and the particular needs of graphical inference. In the interest of helping others to avoid these pitfalls, we use this section to outline our basic approach and to explain why, perhaps counter-intuitively, both classical and recursive IVM techniques are a poor fit for convergent inference on graphical models.

5.3.1 The Algorithm

Our first approach at convergent inference used IVM to compute and iteratively revise an inference query over a progressively larger fraction of the input dataset. That is, we declared the inference query as a materialized view using exactly the query defined in Section 4.2. The set of factor tables was initially empty. As in Cyclic Sampling, we iteratively insert rows of the input factor tables in a shuffled order. A backend IVM system updates the inference result, eventually converging to a correct value once all factor rows have been inserted. This process is summarized in Algorithm 8.

While naive cyclic sampling samples directly from the output of the join, IVM-CIA constructs the same output by iteratively combining parts of the factor tables. The resulting update sequence follows a pattern similar to that of multi-dimensional Ripple Joins [61], incrementally filling the full sample space of the join query. As in naive cyclic sampling, this process may be interrupted at any time to obtain an estimate of $P(\mathcal{Y})$ by taking the already materialized partial result and scaling it by the proportion of samples used to construct it. This proportion can be computed

Algorithm 8 `SimpleIVM-CIA`($\mathcal{B}$, Q)

Require: A bayes net $\mathcal{B}$=($\mathcal{G}(\mathbf{X})$,P)
Require: A conditional probability query: Q=$P(\mathbf{X_q})$
Ensure: The set $ret_{\mathbf{X_q}} = P(\mathbf{X_q})$
1: **for all** $\phi_i \in G(\mathbf{X})$ **do**
2: $index_{0,i} =$ `rand_int`() mod $|dom(X_i)|$
3: $a_i, b_i, m_i \leftarrow$ `InitLCG`($|dom(X_i)|$)
4: $\phi'_i \leftarrow \emptyset$
5: Compile IVM Program Q' to compute $P(\mathbf{X_q})$ from $\{\phi'_i\}$
6: **for all** $k \in 1 \ldots max_i(|dom(X_i)|)$ **do**
7: **for all** $\phi_i \in G(\mathbf{X})$ **do**
8: **if** $k \leq |dom(X_i)|$ **then**
9: $index_{k,i} \leftarrow (a * index_{k-1,i} + b) \mod m_i$
10: Update Q' with row $\phi'_i[index_{k,i}] \leftarrow \phi_i[index_{k,i}]$
11: $ret_{\mathbf{X_q}} \leftarrow$ the output of Q'

by adding a `COUNT(*)` aggregate to each query. IVM-CIA uses the underlying IVM engine to simultaneously track both the estimate and the progress towards an exact result.

5.3.2 Post-Mortem

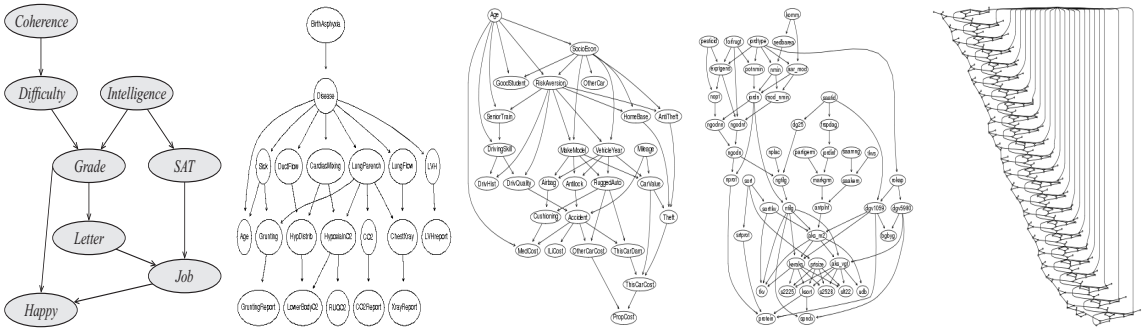
For my first attempt at an IVM-based convergent inference algorithm, we used DBToaster [81], a recursive IVM compiler. DBToaster is aimed at relational data and uses a sparse table encoding that, as we have already mentioned, is ill suited for graphical models. Recognizing this as a bottleneck, we decided to create a modified version of DBToaster that used dense array-based table encodings. Although this optimization did provide a significant speed-up, the resulting engine's performance was still inferior: It converged much slower than variable elimination and had a shallower result quality ramp than the approximation techniques (even cyclic sampling in some cases).

Ultimately, we identified two key features of graphical models that made them ill-suited for database-centric IVM and in particular recursive IVM. First, in Section 5.2,

we noted that nodes in a Bayesian Network tend to have many neighbors and that the network tends to have high hypertree-width. The size of intermediate tables is exponential in the hypertree-width of the query — rather large in the case of graphical models. Recursive IVM systems like DBToaster are in-effect a form of dynamic programming, improving performance by consuming more space. DBToaster in particular maintains materialized copies of intermediate tables for all possible join plans. As one might imagine, the space required for even a BN of moderate size can quickly outpace the available memory.

The second, even more limiting factor of IVM for graphical models is the fan-out of joins. Because of the density of a graphical model's input factors and intermediate tables, joins are frequently not just many-many, but all-all. Thus a single insertion (or batch of insertions) is virtually guaranteed to affect every result row. In recursive IVM, the problem is worse, as each insertion can trigger full-table updates for nearly every intermediate table (which as already noted, can be large). Batching did improve performance, but not significantly enough to warrant replacing cyclic sampling.

Our approach of Leaky Joins was inspired, in large part, by an alternative form of recursive IVM initially described by Ross et. al., [103], which only materializes intermediates for a single join plan. Like this approach, Leaky Joins materializes only a single join plan, propagating changes through the entire query tree. However, unlike the Ross recursive join algorithm, each Leaky Join operator acts as a sort of batching blocker. New row updates are held at each Leaky Join, and only propagated in a random order dictated by the LCG.



(a) Extended Student (b) Child (c) Insurance (d) Barley (e) Diabetes

Figure 5-2: Visualizations of five graphical models from [107] used in our experiments.

5.4 Evaluation

Recall in Section 5.2, we claimed CIAs should satisfy four properties. In this section we present experimental results to show that they do. Specifically, we want to show: (1,2) **Flexibility:** CIAs are able to provide both approximate results and exact results in the inference process. (3) **Approximation Accuracy:** Given the same amount of time, CIAs can provide approximate results with an accuracy that is competitive with state-of-the-art approximation algorithms. (4) **Exact Inference Efficiency:** The time a CIA takes to generate an exact result is competitive with state-of-the-art exact inference algorithms.

5.4.1 Experimental Setup and Data

Experiments were run on a 12 core, 2.5 GHz Intel Xeon with 198 GB of RAM running Ubuntu 16.04.1 LTS. Our experimental code was written in Java and compiled and run single-threaded under the Java HotSpot version 1.8 JDK/JVM. For experiments, we used five probabilistic graphical models from publicly available sources, including the bnlearn Machine Learning Repository [107] to compare the available algorithms. Visualizations of all five graphs are shown in Figure 5-2.

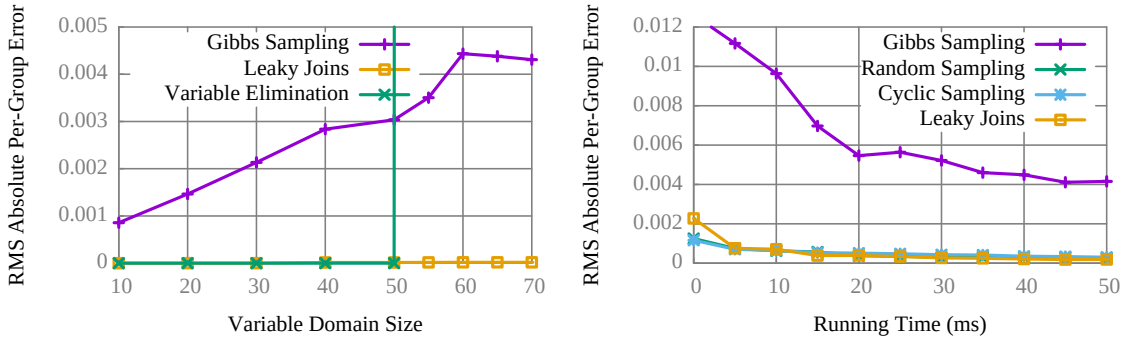
Student. The first data set is the extended version of the Student graphical model from [82]. This graphical model contains 8 random variables. All the random variables are discrete. In order to observe how CIAs are influenced by exponential blowup of scale, we use this graph as a micro-benchmark by generating synthetic factor tables for the Student graph. In the synthetic data, we vary the domain size of each random variable from 2 to 25. Marginals were computed over the `Happiness` attribute.

Child. The second graphical model captures the symptoms and diagnosis of Asphyxia in children [38]. The number of random variables in the graph is 20. All the random variables are discrete with different domain sizes. There are 230 parameters in factors. The average degree of nodes in the graph is 3.5 and the maximum in-degree in the graph is 4. Marginals were computed over the `sick` variable.

Insurance. The third graphical model we used models potential clients for car insurance policies [26]. It contains 27 nodes. All the random variables are discrete with different domain sizes. There are 984 parameters in factors. The average number of degree is 3.85 and the maximum in-degree in the graph is 3. Marginals were computed over the `PropCost` variable.

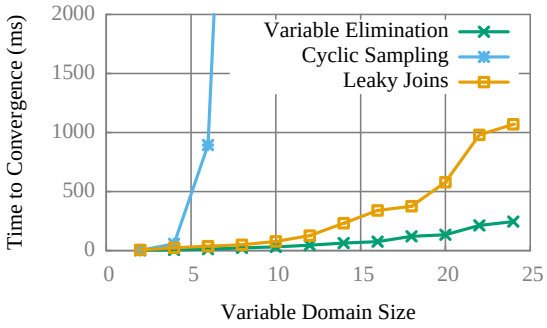
Barley. The fourth graphical model is developed from a decision support system for mechanical weed control in malting barley [84]. The graph contains 48 nodes. All the random variables are discrete with different domain sizes. There are 114005 parameters in factors. The average degree of nodes in the graph is 3.5 and the maximum in-degree in the graph is 4. Marginals were computed over the `ntilg` variable.

Diabetes. The fifth graphical model is a very large graph that captures a model for adjusting insulin [16]. The number of random variables in the graph is 413. All the random variables are discrete with different domain sizes. There are 429409



(a) Accuracy within 10ms

(b) Accuracy over time



(c) Running time to convergence

Figure 5-3: Microbenchmarks on the synthetic, extended Student graph (Figure 5-2a)

parameters in factors. In average, there are 2.92 degrees and the maximum in-degree in the graph is 2. Marginals were computed over the `bg_5` variable.

5.4.2 Inference Methods

We compare our two convergent inference algorithms with variable elimination (for exact inference results) and Gibbs sampling (for approximate inference). We are interested in measuring runtime for exact inference and accuracy for approximate inference. We assign an index for each node in the Bayes net following the topological order. We assume that for each factor ϕ_i, the variables are ordered by the following conventions: $pa(\mathbf{X}) \prec \mathbf{X}$, where $\mathbf{X}=\{X_i, \ldots, X_j, \ldots\}$ is ordered increasingly by index. The domain values in each random variable is ordered increasingly.

Variable Elimination. Variable elimination is the classic exact inference algo-

rithm used for graphical models, as detailed in Section 4. As is standard in variable elimination, intermediate join results are streamed and never actually materialized. To decide on an elimination (join) ordering, we adopt the heuristic methods in [82]. At each point, the algorithm evaluates each of the remaining variables in the Bayes net based on a heuristic cost function. The cost criteria used for evaluating each variable is Min-weight, where the cost of a node is the product of weights, where weight is the domain cardinality of the node's neighbors. For example, using Min-weight, the selected order for the extended student graph in Figure 5-2a is: C, D, S, I, L, J, G. H is the target random variable.

Gibbs Sampling. As discussed in Section 4, Gibbs sampling first generates the initial sample with each variable X_i in $\mathcal{B} = (\mathcal{G}(\mathbf{X}), \Phi)$ following the conditional probability distribution $p(X_i|pa(X_i))$. Then, we randomly fix the value for some random variable X_j and use it as evidence to generate next sample. With more and more samples collect, the distribution will get increasingly closer to the posterior. We skip the first hundred samples for small graphs 5-2a, 5-2b and 5-2c, and thousand samples for larger graphs 5-2d and 5-2e at the beginning of the sample process. The target probability distribution is calculated by normalizing the sum of sample frequencies for each value in the target variables X_q.

Cyclic Sampling. Cyclic Sampling is our first CIA, described in Section 5.2.1 and in Algorithm 6. Note that cyclic sampling does not materialize the joint probability distribution, but rather constructs rows of the joint distribution dynamically using a LCG (Equation 5.1). This process takes constant time for a fixed graph.

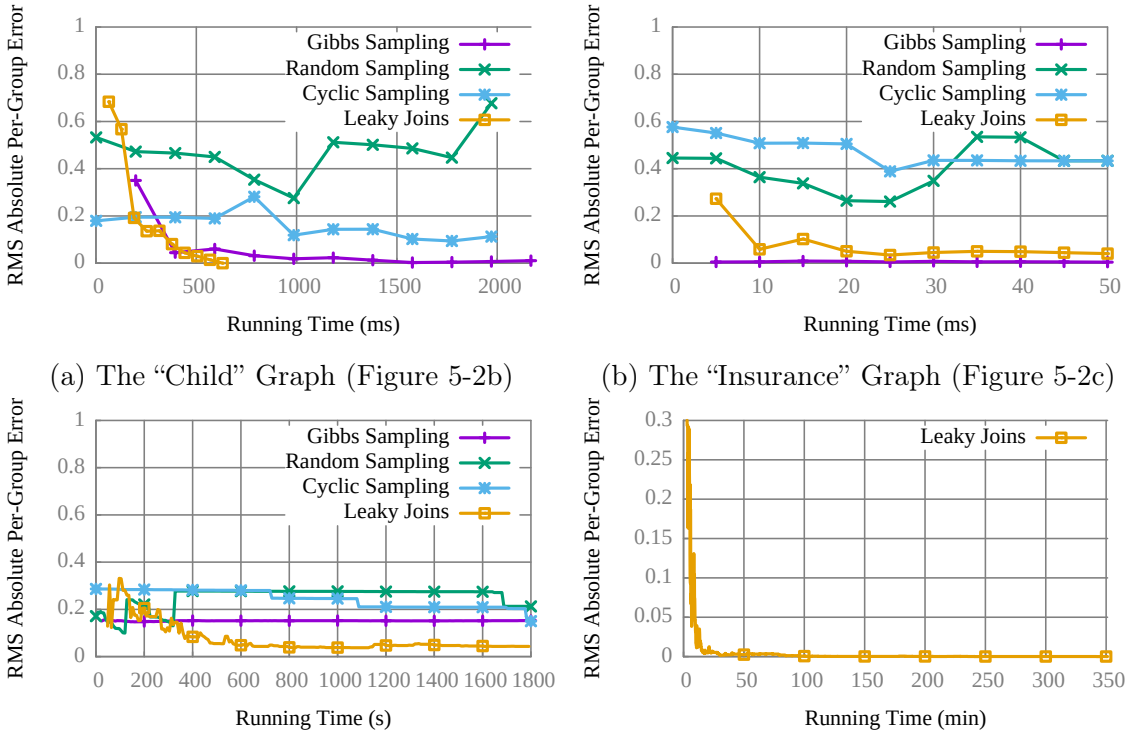
Leaky Joins. Leaky Joins, our second CIA, were described in Section 5.2.2. We use the same elimination (join) order as in variable elimination algorithm to construct the clique tree and materialize intermediate tables, Clique's clusters $\mathbf{C}$ and Clique's

seperator **S**. Then we conduct the “passing partial messages" process according to Algorithm 7.

5.4.3 Flexibility

We first explore the flexibility of CIAs by comparing the accuracy of different algorithms (both exact and approximate) within a finite cutoff time. We imitate the situation that time is the major concern for user and the goal is to provide an accurate inference result at a given time. We compute the marginal probability, cutting each algorithm off after the predefined period, and average the fractional error across each marginal “group”.

Figure 5-3a shows the average fractional error for each inference algorithm on the **student** graph with a cutoff of 10 seconds. We vary the factor size from 10 to 70 to simulate small and large graphs. Variable elimination provides an exact inference result for variables with domain size smaller than 50. On larger graphs, it times out, resulting in a 100% error. Gibbs sampling can always provide an approximate result, but produces results that are inaccurate, even when variable elimination can produce an exact result. Leaky joins provide exact inference results when the domain size is smaller than 40, but when the domain size passes 40, it still provides approximate results with a lower error than Gibbs sampling. This graph shows that, with leaky joins, the same algorithm can support both the exact inference and approximate inference cases; neither users or inference engines need to anticipate which class of algorithms to use.



(a) The "Child" Graph (Figure 5-2b)

(b) The "Insurance" Graph (Figure 5-2c)

(c) The "Barley" Graph (Figure 5-2d)

(d) The "Diabetes" Graph (Figure 5-2e)

Figure 5-4: Approximation accuracy for real-world graphs

5.4.4 Approximate Inference Accuracy

Figure 5-3b compares each algorithm's approximate accuracy relative to time spent on the **student** graph with a node size of 35. At each time t, the average fractional error of leaky joins is smaller than that of Gibbs sampling. In addition, leaky join converges to exact result, which Gibbs sampling will never do. The steep initial error in leaky joins at the start stems from the first few sampling rounds for each intermediate table C_i being based on weak preliminary approximations; The algorithm needs some burn-in time to have samples to cover their domains to provide approximate results. Gibbs sampling algorithm also has burn-in process. The sharp curve for Gibbs sampling is because it takes more samples for Gibbs sampling to cover corner cases, and generate samples with less probabilities. Figure 5-4a, Figure 5-4b and Figure 5-4c show the approximate accuracy result for child and insurance graph. Gibbs sampling performs

well in this two graph for that the domain of the target variables X_q are small (the domain of sick node for child graph is two and propcost for insurance is 4). There will be less corner cases and by *Chernoff's bound* [35], the number of samples to required decreases as the probability of $\mathrm{P}(X_q)$ increases.

$$P_{est}(P_{est}(X_q) \notin P(X_q)(1 \pm \epsilon)) \leq 2e^{-NP(X_q)\epsilon^2/3} \leq \delta.$$

The result shows that even in this situation, the approximate result of leaky joins is still comparable to Gibbs sampling. Of these four graphs, the "Diabetes" graph was the most complex, and only Leaky Joins produced meaningful results.

For comparison with the accuracy results in Figure 5-4, Variable Elimination produces exact results for "Child" in 19 ms, for "Insurance" in 49 ms, for Barley in approximately 1.4 hours, and for Diabetes in approximately 1.5 hours.

5.4.5 Convergence Time

Figure 5-3c shows the exact running time for variable elimination and leaky joins. Recall that the running complexity of variable elimination and leaky join is dominated by the size of the clique's cluster, $O(k|C_{max}|)$, where $|C_{max}|$ is the size of largest clique's cluster. The difference is that leaky join has a constant k. As the factor size increases, C_{max} increases and both algorithms get slower at equivalent rates.

5.4.6 Memory

Unlike variable elimination and many of the approximate algorithms, leaky joins does continuously maintain materialized intermediate results. To measure memory use, we used the JVM's `Runtime.totalMemory()` method, sampling immediately after results were produced, but before garbage collection could be run. Figure 5-5 shows

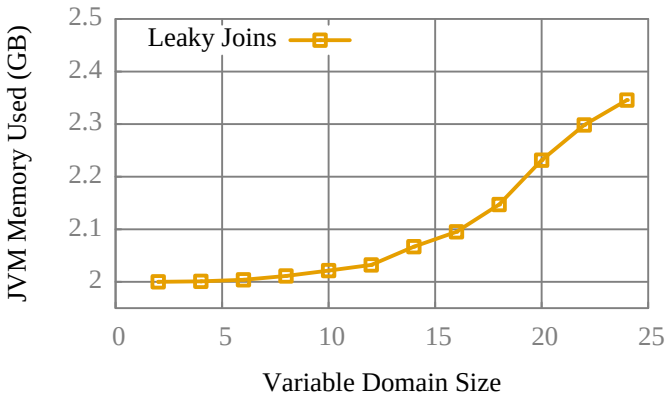


Figure 5-5: JVM Memory use for Cyclic Sampling. Note that at startup, Java has already allocated roughly 2 GB.

Java's memory usage with leaky joins for the "Student" micro benchmark. The actual memory needs of leaky joins are comparatively small: The two largest intermediate results have roughly 20-thousand rows, with 5 columns each. Overall, Leaky Joins only forms a small portion of Java's overall footprint.

5.5 CIA Applications

As motivated in Chapter 1, inferene in probabilistic graphical model (PGM) has a lot of application in many domains. Since convergent inference algorithms are a special kind of inference algorithms, they can be applied to these areas as well. In this section, I discuss how CIAs are used in data cleaning as Domain Constraint Repair lens, and its application as a analysis service in Cloud service.

5.5.1 CIA Domain Constraint Repair Lens

As discussed in Section 3.1, generally speaking, all kinds of algorithms can be applied as the model in Domain Constraint Repain lens, as long as the algorithm provides a probability distribution of the querying variable as result. CIA satisfies this condition. Moreover, CIA can provide the trade-off between computation time and accuracy and

provides quality of current proabability distribution result to help improve ranking. In Example 4 in Chapter 3.1, Alice creates a lens to handle missing values in Product table, she can use CIA to help her model missing attributes *category* and textitbrand.

```
CREATE LENS SaneProduct AS SELECT * FROM Product
USING DOMAIN_REPAIR( category string NOT NULL,
                     brand string NOT NULL );
```

Behind the scence, Bayesian Network of product is constructed from the certain data in Product table, For product with id P123 in Figure 1-1, *brand*'s value is missing, in CIA, it means we are trying to find P(*brand*|id=P123,name="Apple 6s, White",category="phone"). Furthermore, for row with id P2345 in Product table, if cleaning *category* and *brand* can be done together, we can model the joint probability distribution P(*category*, *brand*|id=P2345,name="Sony to inches"), which will be more accurate than calculating the probability independently since they may be correlated.

One advantage of CIA in Domain Constraint Repair lens is that user can control the execution time and have an idea how uncertain current probability distribution is. Since as shown in Section 3.3, the ranking of data cleaning task is based on the confidence we have on the attribute's value based on its probability distribution (Using entropy). Therefore, the more accurate the probability distribution is, the more accurate ranking of cleaning task we will have. Therefore, when we use entropy or information gain to rank the cleaning task, considering the quallity of probability distribution result (by $\epsilon - \delta$ error bounds) will improve accuracy.

5.5.2 Flexible Pricing Plan Data Analysis systems on Cloud Service

A lot of data analysis is done using cloud services. The cloud service providers usually provide computing instances with data analysis tools pre-installed in it such as Amazon EMR and Windows Azuew HDInsight. Current pricing plans for these services is: user pay an hourly rate for every instance hour they use. This rate depends on the instance type, for example, standard, high CPU, high memory, high storage and so on. For a normal user, especially who is not an expert in machine learning, choosing a cost efficiently plan is difficult. For inference in PGM, with CIA, the cloud service can provide new pricing plans can free user from the difficulty of choosing the proper computing instances. There are two forms of pricing plans CIA can provide: `(1) Running time vs Pricing.` After user upload the their data into cloud and submitting their query, we first calculates different running time with according price. Since this running time is based on estimation and theoretical bound, it may not be acurate when the job is acurately running in the cloud. Since CIA provides progress information during the calculation, the cloud service can increase or decrease computing instance accordingly. `(2) On-Demand interactive Pricing plan.` There are also condition that user don't have an idea of cost of his job. He can use a pay-as-you-go way to run his job. First, the inference job is started in standard instance, CIA provides user the current probability distribution and accuracy along with the cost spent so far. If user is satisfied with current accuracy, he can stop the job to save money. Also, CIA can suggest user if he wants speed up, he can spend X dollars to achieve two times speed-up. This suggestion will be very accurate because since the job is already running, the estimation based on current the information collected will be very accurate.

There is existing work on new pricing plan for cloud service and data analytics with performance guarantees [96]. Although there is similarity between this work and my thesis, the difference is clear. While existing work is focusing on general SQL-like data analysis, we are focusing on inference on PGM, Dynamic Bayesian Network including Hidden Markov Model and Conditional Random Field.

Chapter 6

Conclusion and Future Work

In this dissertation, I focus on providing interaction to user during data cleaning and data analysis process.

For interactive data cleaning, I focus on building a system that support On-Demand ETL process - Mimir. Mimir enables composable non-deterministic data processing operators called Lenses that provide the illusion of fully cleaned relational data that can be queried using standard SQL. The composable structure lenses use PC-Tables to encode output, and can be deployed in traditional, deterministic database environments using Virtual C-Tables. Mimir supports best-effort guesses at the contents a PC-Table, evaluation of quality measures over a PC-Table, and a family of heuristics for prioritizing curation tasks called CPI. I have demonstrated the feasibility and need for On-Demand ETL, and the effectiveness of CPI-based heuristics.

In future work on Mimir, I shall continue to focus on providing an On-Demand ETL system that conducts data cleaning in a cost efficient fashion through interaction. With this goal in mind, I describe possible directions as future work for Mimir.

`Lens Composition Optimization.` Since input data usually contains more than one kind of cleaning problems, multiple lenses are used to model them. We need

to comopse these lenses in order to rank the cleaning tasks for feedback. As discussed in Chapter 3.1.3, current lenses are created in a cascaded order. The drawback for this pipelined composition is that errors from one lens will propagate to another. In future, I plan to seek joint modeling solutions for lenses composition. There are two possible directions. The first one is Expectation-Maximization (EM). EM can be used to avoid error propagation. Unlike composition in a pipelined fashion, EM conducts composition as an iterative loop. Each lens serves as the input for each other and forms a loop. This looping process terminates when there is no change of any output or after several iterations. There is existing work using EM to estimate the trustworthiness of web sources [46]. My hope is that EM can be used to compose lenses to avoid error propogation. The second potential direction is to study how to compose lens based on SATsolver [106], which constructs a Markov Logic Network, where each lens is represented as hard constraints. Prior arts have already show potential solutions such as WalkSAT and MaxWalkSAT implemented in Alchemy [45] and Tuffy [92]. I hope to construct a Markov Logic Network to simulate all lenses.

For interactive data analysis, I introduced a class of convergent inference algorithms (CIAs) based on sampling without replacement using linear congruential generators. I proposed CIAs built over incremental view maintenance and a novel aggregate join algorithm that I call Leaky Joins. I evaluated both IVM-CIA and Leaky Joins, and found that Leaky Joins were able to approximate the performance of Variable Elimination on simple graphs, and the accuracy of state-of-the-art approximation techniques on complex graphs. As graph complexity increased, the bounded-time accuracy of Leaky Joins degraded gracefully.

My algorithms has one limitation which represents opportunities for future work. My algorithm didn't consider scalability. In the era of Big Data, distributed inference in graphical model is necessary for performance. Similar to works for parallelizing

existing inference algorithms [19, 41], our algorithm is possible to run in parallel.

Bibliography

[1] Bestbuy api. http://developer.bestbuy.com/documentation/products-api. Accessed: 2017-04-26.

[2] Data scientists survey. https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#54d556dd6f63. Accessed: 2017-04-26.

[3] Ebay api. http://go.developer.ebay.com. Accessed: 2017-04-26.

[4] Real estate data. http://pages.cs.wisc.edu/~anhai/wisc-si-archive/domains/real_estate1.html. Accessed: 2017-04-26.

[5] Walmartlabs api. http://developer.walmartlabs.com/docs/read/Search_API. Accessed: 2017-04-26.

[6] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. The Aqua approximate query answering system. *SIGMOD Rec.*, 28(2):574–576, 1999.

[7] Sameer Agarwal, Anand P. Iyer, Aurojit Panda, Samuel Madden, Barzan Mozafari, and Ion Stoica. Blink and it's done: Interactive queries on very large data. *pVLDB*, 5(12):1902–1905, 2012.

[8] Sameer Agarwal, Aurojit Panda, Barzan Mozafari, Samuel Madden, and Ion Stoica. BlinkDB: Queries with bounded errors and bounded response times on very large data. Technical report, ArXiV, 03 2012.

[9] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha U. Nabar, Tomoe Sugihara, and Jennifer Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, pages 1151–1154. ACM, 2006.

[10] Yanif Ahmad, Oliver Kennedy, Christoph Koch, and Milos Nikolic. DBToaster: Higher-order delta processing for dynamic, frequently fresh views. *pVLDB*, 5(10):968–979, 2012.

[11] Bogdan Alexe, Laura Chiticariu, Renée J. Miller, and Wang Chiew Tan. Muse: Mapping understanding and design by example. In *ICDE*, pages 10–19. IEEE, 2008.

[12] Sarah R. Allen, Lisa Hellerstein, Devorah Kletenik, and Tonguç Ünlüyurt. Evaluation of DNF formulas. In *ISAIM*, 2014.

[13] Yasser Altowim, Dmitri V. Kalashnikov, and Sharad Mehrotra. Progressive approach to relational entity resolution. *PVLDB*, 7(11):999–1010, 2014.

[14] Hotham Altwaijry, Dmitri V. Kalashnikov, and Sharad Mehrotra. Query-driven approach to entity resolution. *Proc. VLDB Endow.*, 6(14):1846–1857, September 2013.

[15] Yael Amsterdamer, Susan B. Davidson, Tova Milo, Slava Novgorodov, and Amit Somech. OASSIS: query driven crowd mining. In *SIGMOD*, pages 589–600. ACM, 2014.

[16] Steen Andreassen, Roman Hovorka, Jonathan Benn, Kristian G Olesen, and Ewart R Carson. A model-based approach to insulin adjustment. In *AIME 91*, pages 239–248. Springer, 1991.

[17] Lyublena Antova, Christoph Koch, and Dan Olteanu. $10^{(10^6)}$ worlds and beyond: efficient representation and processing of incomplete information. *VLDB J.*, 18(5):1021–1040, 2009.

[18] Subi Arumugam, Fei Xu, Ravi Jampani, Christopher Jermaine, Luis L. Perez, and Peter J. Haas. MCDB-R: Risk analysis in the database. *pVLDB*, 3(1-2):782–793, 2010.

[19] Ron Bekkerman, Mikhail Bilenko, and John Langford. *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press, 2011.

[20] Khalid Belhajjame, Norman W. Paton, Suzanne M. Embury, Alvaro A. A. Fernandes, and Cornelia Hedeler. Feedback-based annotation, selection and refinement of schema mappings for dataspaces. In *EDBT*, volume 426, pages 573–584. ACM, 2010.

[21] Khalid Belhajjame, Norman W. Paton, Alvaro A. A. Fernandes, Cornelia Hedeler, and Suzanne M. Embury. User feedback as a first class citizen in information integration systems. In *CIDR*, pages 175–183, 2011.

[22] Zohra Bellahsene, Angela Bonifati, and Erhard Rahm, editors. *Schema Matching and Mapping*. Data-Centric Systems and APPLICATIONS. Springer, 2011.

[23] Philip A. Bernstein, Jayant Madhavan, and Erhard Rahm. Generic schema matching, ten years later. *PVLDB*, 4(11):695–701, 2011.

[24] Philip A. Bernstein, Sergey Melnik, and John E. Churchill. Incremental schema matching. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pages 1167–1170. VLDB Endowment, 2006.

[25] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *JMLR*, 11:1601–1604, 2010.

[26] John Binder, Daphne Koller, Stuart Russell, and Keiji Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29(2-3):213–244, 1997.

[27] Philippe Bonnet and Anthony Tomasic. Partial answers for unavailable data sources. In *FQAS*, volume 1495, pages 43–54. Springer, 1998.

[28] Jihad Boulos, Nilesh N. Dalvi, Bhushan Mandhani, Shobhit Mathur, Christopher Ré, and Dan Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *SIGMOD*, pages 891–893. ACM, 2005.

[29] Héctor Corrada Bravo and et al. Optimizing mpf queries: Decision support and probabilistic inference, 2007.

[30] Huiping Cao, Yan Qi, K. Selçuk Candan, and Maria Luisa Sapino. Feedback-driven result ranking and query refinement for exploring semi-structured data collections. In *EDBT*, volume 426, pages 3–14. ACM, 2010.

[31] Xiaoyong Chai, Ba-Quy Vuong, AnHai Doan, and Jeffrey F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 87–100, New York, NY, USA, 2009. ACM.

[32] Xiaoyong Chai, Ba-Quy Vuong, AnHai Doan, and Jeffrey F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In *SIGMOD*, pages 87–100. ACM, 2009.

[33] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 313–324, New York, NY, USA, 2003. ACM.

[34] Surajit Chaudhuri and Kyuseok Shim. Including group-by in query optimization. In *VLDB*, 1994.

[35] Herman Chernoff. A career in statistics. *Past, Present, and Future of Statistical Science*, page 29, 2014.

[36] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2201–2206. ACM, 2016.

[37] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1247–1261. ACM, 2015.

[38] A Philip Dawid. Prequential analysis, stochastic complexity and bayesian inference. *Bayesian statistics*, 4:109–125, 1992.

[39] Amol Deshpande, Lisa Hellerstein, and Devorah Kletenik. Approximation algorithms for stochastic boolean function evaluation and stochastic submodular set cover. In *SODA*, pages 1453–1467. SIAM, 2014.

[40] Amol Deshpande and Samuel Madden. MauveDB: Supporting model-based user views in database systems. In *SIGMOD*, 2006.

[41] FJ Diez and José Mira. Distributed inference in bayesian networks. *Cybernetics and Systems: An International Journal*, 25(1):39–61, 1994.

[42] Hong Hai Do and Erhard Rahm. COMA - A system for flexible combination of schema matching approaches. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pages 610–621, 2002.

[43] AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. *SIGMOD Rec.*, 30(2):509–520, May 2001.

[44] Alin Dobra, Chris Jermaine, Florin Rusu, and Fei Xu. Turbo-charging estimate convergence in DBO. *pVLDB*, 2(1):419–430, August 2009.

[45] Pedro Domingos and Daniel Lowd. Markov logic: An interface layer for artificial intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 3(1):1–155, 2009.

[46] Xin Luna Dong, Evgeniy Gabrilovich, Kevin Murphy, Van Dang, Wilko Horn, Camillo Lugaresi, Shaohua Sun, and Wei Zhang. Knowledge-based trust: Estimating the trustworthiness of web sources. *Proceedings of the VLDB Endowment*, 8(9):938–949, 2015.

[47] Xin Luna Dong, Alon Halevy, and Cong Yu. Data integration with uncertainty. *The VLDB Journal*, 18(2):469–500, April 2009.

[48] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE TKDE*, 19(1):1–16, January 2007.

[49] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: Getting to the core. In *PODS*, pages 90–101. ACM, 2003.

[50] Robert Fink, Andrew Hogue, Dan Olteanu, and Swaroop Rath. Sprout^2: a squared query engine for uncertain web data. In *SIGMOD*, pages 1299–1302. ACM, 2011.

[51] Alberto Freitas, Altamiro Costa-Pereira, and Pavel Brazdil. Cost-sensitive decision trees applied to medical data. In *DaWaK*, pages 303–312. 2007.

[52] Amélie Gheerbrant, Leonid Libkin, and Cristina Sirangelo. When is naive evaluation possible? In *PODS*, pages 75–86. ACM, 2013.

[53] Stephan F Gohmann, Robert M Barker, David J Faulds, and Jian Guan. Salesforce automation, perceived information accuracy and user satisfaction. *Journal of Business & Industrial Marketing*, 20(1):23–32, 2005.

[54] Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Provenance in ORCHESTRA. *DEBU*, 33(3):9–16, 2010.

[55] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40. ACM, 2007.

[56] Todd J. Green and Val Tannen. Models for incomplete and probabilistic information. *IEEE Data Eng. Bull.*, 29(1):17–24, 2006.

[57] Anja Gruenheid, Xin Luna Dong, and Divesh Srivastava. Incremental record linkage. *PVLDB*, 7(9):697–708, 2014.

[58] Chenjuan Guo, Cornelia Hedeler, Norman W. Paton, and Alvaro A. A. Fernandes. Matchbench: Benchmarking schema matching algorithms for schematic correspondences. In *Big Data - 29th British National Conference on Databases, BNCOD 2013, Oxford, UK, July 8-10, 2013. Proceedings*, pages 92–106, 2013.

[59] Xintong Guo, Hongzhi Wang, Yangqiu Song, and Gao Hong. Brief survey of crowdsourcing for data mining. *ESWA*, 41(17):7987–7994, 2014.

[60] Rahul Gupta and Sunita Sarawagi. Creating probabilistic databases from information extraction models. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pages 965–976. VLDB Endowment, 2006.

[61] Peter J. Haas and Joseph M. Hellerstein. Ripple joins for online aggregation. In *SIGMOD*, 1999.

[62] P.J. Haas. Large-sample and deterministic confidence intervals for online aggregation. In *SSDBM*, pages 51–62, 1997.

[63] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online aggregation. *SIGMOD Rec.*, 26(2):171–182, 1997.

[64] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online aggregation. In *SIGMOD*, pages 171–182, 1997.

[65] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *JASS*, 58(301):13–30, 1963.

[66] Zhen Hua Liu and Dieter Gawlick. Management of flexible schema data in RDBMSs - opportunities and limitations for NoSQL. In *CIDR*, 2015.

[67] Jiewen Huang, Lyublena Antova, Christoph Koch, and Dan Olteanu. MayBMS: a probabilistic database management system. In *SIGMOD*, pages 1071–1074. ACM, 2009.

[68] Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[69] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher Jermaine, and Peter J. Haas. MCDB: A monte carlo approach to managing uncertain data. In *SIGMOD*, 2008.

[70] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher Jermaine, and Peter J Haas. MCDB: a monte carlo approach to managing uncertain data. In *SIGMOD*, pages 687–700, 2008.

[71] Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD*, pages 847–860. ACM, 2008.

[72] Chris Jermaine, Subramanian Arumugam, Abhijit Pol, and Alin Dobra. Scalable approximate query processing with the DBO engine. *TODS*, 33(4):23:1–23:54, 2008.

[73] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive visual specification of data transformation scripts. In *ACM Human Factors in Computing Systems (CHI)*, 2011.

[74] Haim Kaplan, Eyal Kushilevitz, and Yishay Mansour. Learning with attribute costs. In *STOC*, pages 356–365, 2005.

[75] Oliver Kennedy and Christoph Koch. PIP: A database system for great and small expectations. In *ICDE*, pages 157–168, 2010.

[76] Oliver Kennedy and Suman Nath. Jigsaw: Efficient optimization over uncertain enterprise data. In *SIGMOD*, 2011.

[77] Oliver Kennedy and Suman Nath. Jigsaw: efficient optimization over uncertain enterprise data. In *SIGMOD*, pages 829–840. ACM, 2011.

[78] Oliver Kennedy, Ying Yang, Jan Chomicki, Ronny Fehling, ZhenHua Liu, and Dieter Gawlick. Detecting the temporal context of queries. In *BIRTE*, volume 206 of *LNBIP*, pages 97–113. 2015.

[79] Anja Klein, Rainer Gemulla, Philipp Rösch, and Wolfgang Lehner. Derby/S: A DBMS for sample-based query answering. In *SIGMOD*, 2006.

[80] Donald Knuth. The art of computer programming. semi-numerical algorithms. 1968.

[81] Christoph Koch, Yanif Ahmad, Oliver Kennedy, Milos Nikolic, Andres Nötzli, Daniel Lupei, and Amir Shaikhha. DBToaster: higher-order delta processing for dynamic, frequently fresh views. *VLDBJ*, 23(2):253–278, 2014.

[82] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[83] Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proc. VLDB Endow.*, 3(1-2):484–493, September 2010.

[84] Kristian Kristensen and I Rasmussen. A decision support system for mechanical weed control in malting barley. In *ECITA*, 1997.

[85] Willis Lang, Rimma V. Nehme, Eric Robinson, and Jeffrey F. Naughton. Partial results in database systems. In *SIGMOD*, pages 1275–1286. ACM, 2014.

[86] Pierre L'Ecuyer. Random numbers for simulation. *CACM*, 33(10):85–97, 1990.

[87] Yoonkyong Lee, Mayssam Sayyadian, AnHai Doan, and Arnon Rosenthal. etuner: tuning schema matching software using synthetic scenarios. *VLDB J.*, 16(1):97–122, 2007.

[88] M. Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013.

[89] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 49–58, 2001.

[90] Yosi Mass, Maya Ramanath, Yehoshua Sagiv, and Gerhard Weikum. IQ: the case for iterative querying for knowledge. In *CIDR*, pages 38–44, 2011.

[91] Chris Mayfield, Jennifer Neville, and Sunil Prabhakar. Eracer: A database approach for statistical inference and data cleaning. In *SIGMOD*, 2010.

[92] Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *Proceedings of the VLDB Endowment*, 4(6):373–384, 2011.

[93] Marlon Núñez. The use of background knowledge in decision tree induction. *JMLR*, 6:231–250, 1991.

[94] Frank Olken and Doron Rotem. Simple random sampling from relational databases. In *VLDB*, 1986.

[95] Dan Olteanu, Jiewen Huang, and Christoph Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, pages 145–156. IEEE, 2010.

[96] Jennifer Ortiz, Brendan Lee, and Magdalena Balazinska. Perfenforce demonstration: Data analytics with performance guarantees. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2141–2144. ACM, 2016.

[97] F. Panse, M. van Keulen, and N. Ritter. Indeterministic handling of uncertain decisions in deduplication. *Journal of Data and Information Quality*, 4(2):9, March 2013.

[98] Stephen K. Park and Keith W. Miller. Random number generators: good ones are hard to find. *CACM*, 31(10):1192–1201, 1988.

[99] Eric Peukert, Henrike Berthold, and Erhard Rahm. Rewrite techniques for performance optimization of schema matching processes. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 453–464, New York, NY, USA, 2010. ACM.

[100] Erhard Rahm and Philip A Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[101] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[102] Vijayshankar Raman and Joseph M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 381–390, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[103] Kenneth A. Ross, Divesh Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. *SIGMOD Rec.*, 25(2):447–458, 1996.

[104] Florin Rusu and Alin Dobra. Glade: A scalable framework for efficient analytics. *OSR*, 46(1):12–18, February 2012.

[105] Bruce Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. Wiley & Sons, 2007.

[106] Uwe Schöning and Jacobo Torán. *The Satisfiability Problem: Algorithms and Analyses*, volume 3. Lehmanns Media, 2013.

[107] Marco Scutari. The bnlearn bayesian network repository. http://www.bnlearn.com/bnrepository/.

[108] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Prdb: Managing and exploiting rich correlations in probabilistic databases. *VLDB Journal, special issue on uncertain and probabilistic databases*, 2009.

[109] Robert J Serfling. Probability inequalities for the sum in sampling without replacement. *The Annals of Statistics*, pages 39–48, 1974.

[110] Frank L Severence. *System modeling and simulation: an introduction.* John Wiley & Sons, 2009.

[111] Sarvjeet Singh, Chris Mayfield, Sagar Mittal, Sunil Prabhakar, Susanne Hambrusch, and Rahul Shah. Orion 2.0: Native support for uncertain data. In *SIGMOD*, pages 1239–1242. ACM, 2008.

[112] Parag Singla and Pedro Domingos. Entity resolution with markov logic. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*, pages 572–582, 2006.

[113] Michael Stonebraker, Paul Brown, Alex Poliakov, and Suchi Raman. *The Architecture of SciDB.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[114] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. Probabilistic databases. *Synthesis Lectures on Data Management*, 3(2):1–180, 2011.

[115] Ming Tan and Jeffrey C. Schlimmer. Cost-sensitive concept learning of sensor use in approach and recognition. In *MLSB*, 1989.

[116] Tonguç Ünlüyurt. Sequential testing of complex systems: a review. *DAM*, 142(1-3):189–205, 2004.

[117] Susan V. Vrbsky and Jane W.-S. Liu. APPROXIMATE - A query processor that produces monotonically improving approximate answers. *IEEE TKDE*, 5(6):1056–1068, 1993.

[118] Daisy Zhe Wang, Yang Chen, Christan Earl Grant, and Kun Li. Efficient in-database analytics with graphical models. *IEEE Data Eng. Bull.*, 37(3):41–51, 2014.

[119] Daisy Zhe Wang, Michael J. Franklin, Minos Garofalakis, and Joseph M. Hellerstein. Querying probabilistic information extraction. *Proc. VLDB Endow.*, 3(1-2):1057–1067, September 2010.

[120] Daisy Zhe Wang, Michael J. Franklin, Minos Garofalakis, Joseph M. Hellerstein, and Michael L. Wick. Hybrid in-database inference for declarative information extraction. In *SIGMOD*, 2011.

[121] Daisy Zhe Wang, Michael J. Franklin, Minos N. Garofalakis, Joseph M. Hellerstein, and Michael L. Wick. Hybrid in-database inference for declarative information extraction. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pages 517–528, 2011.

[122] Daisy Zhe Wang, Eirinaios Michelakis, Michael J. Franklin, Minos N. Garofalakis, and Joseph M. Hellerstein. Probabilistic declarative information extraction. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 173–176, 2010.

[123] Daisy Zhe Wang, Eirinaios Michelakis, Minos Garofalakis, and Joseph M. Hellerstein. Bayesstore: Managing large, uncertain data repositories with probabilistic graphical models. *Proc. VLDB Endow.*, 1(1):340–351, August 2008.

[124] Michael Wick, Andrew McCallum, and Gerome Miklau. Scalable probabilistic databases with factor graphs and mcmc. *pVLDB*, 3(1-2):794–804, 2010.

[125] Michael L. Wick, Andrew McCallum, and Gerome Miklau. Scalable probabilistic databases with factor graphs and MCMC. *CoRR*, abs/1005.1934, 2010.

[126] Pierre Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, (110–111):119–136, 1985.

[127] Sen Wu, Ce Zhang, Feiran Wang, and Christopher Ré. Incremental knowledge base construction using deepdive. *CoRR*, abs/1502.00731, 2015.

[128] Ying Yang. On-demand query result cleaning. In *VLDB PhD Workshop*, 2014.

[129] Chen Jason Zhang, Lei Chen, H. V. Jagadish, and Chen Caleb Cao. Reducing uncertainty of schema matching via crowdsourcing. *Proc. VLDB Endow.*, 6(9):757–768, July 2013.