Managing Large-scale Probabilistic Databases

Christopher Ré

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

Doctor of Philosophy

University of Washington

2009

Program Authorized to Offer Degree: Computer Science & Engineering

University of Washington
Graduate School

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

Christopher Ré

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

Chair of the Supervisory Committee:

_____

Dan Suciu

Reading Committee:

_____

Magdalena Balazinska

_____

Anna Karlin

_____

Dan Suciu

Date: _____

Signature_____

Date_____

University of Washington

Abstract

Managing Large-scale Probabilistic Databases

Christopher Ré

Chair of the Supervisory Committee:
Professor Dan Suciu
Computer Science & Engineering

Modern applications are driven by data, and increasingly the data driving these applications are imprecise. The set of applications that generate imprecise data is diverse: In sensor database applications, the goal is to measure some aspect of the physical world (such as temperature in a region or a person's location). Such an application has no choice but to deal with imprecision, as measuring the physical world is inherently imprecise. In data integration, consider two databases that refer to the same set of real-world entities, but the way in which they refer to those entities is slightly different. For example, one database may contain an entity 'J. Smith' while the second database refers to 'John Smith' . In such a scenario, the large size of the data makes it too costly to manually reconcile all references in the two databases. To lower the cost of integration, state-of-the-art approaches allow the data to be imprecise. In addition to applications which are forced to cope with imprecision, emerging data-driven applications, such as large-scale information extraction, natively produce and manipulate similarity scores. In all these domains, the current state-of-the-art approach is to allow the data to be imprecise and to shift the burden of coping with imprecision to applications. The thesis of this work is that it is possible to effectively manage large, imprecise databases using a generic approach based on probability theory. The key technical challenge in building such a general-purpose approach is performance, and the technical contributions of this dissertation are techniques for efficient evaluation over probabilistic databases. In particular, we demonstrate that it is possible to run complex SQL queries on tens of gigabytes of probabilistic data with performance that is comparable to a standard relational database engine.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I am forever indebted to Dan Suciu. Dan is blessed with an astonishing combination of brilliance and patience, and I will be forever grateful for the skills he has patiently taught me. I owe him my research career, which is one of the most rewarding pursuits in my life.

Magdalena Balazinska offered me patient support and advice throughout my time at the University of Washington. I am also profoundly grateful for her trust that my only vaguely defined research directions would lead to something interesting. I am also grateful to Anna Karlin for her participation on my committee.

The students in the database group and the Computer Science & Engineering department made my graduate studies both intellectually stimulating and incredibly enjoyable. An incomplete list of those who helped me on this path include Eytan Adar, Michael Cafarella, Nilesh Dalvi, YongChul Kwon, Julie Letchner, Jayant Madhavan, Gerome Miklau, Kate Moore, Vibhor Rastogi, and Atri Rudra. The conversations that I had with each one of them enriched my life and made this work a joy to pursue. The faculty and staff at the University of Washington have created a wonderful place, and I am very thankful to be a part of it.

Finally, I would like to thank my family. My father, Donald Ré, not only provided love and support thoughout my life, but also listened to hours of excited chatter about my research ideas that were – at best – only halfway clear in my own mind. John and Peggy Emery provided incredible support including countless meals, rides, and hours of great conversation. I am, however, most grateful to them for two things: their love and the greatest gift of all, their daughter and my wife, Mikel. Mikel's patience, love, and support are boundless. For any current or future success that I might have, Mikel deserves more credit than I can possibly express in a few lines of text.

Chapter 1

INTRODUCTION

A probabilistic database is a general framework for managing imprecision and uncertainty in data. Building a general framework to manage uncertainty in data is a challenging goal because critical data-management tasks, such as query processing, are theoretically and practically more expensive than in traditional database systems. As a result, a major challenge in probabilistic databases is to efficiently query and understand large collections of uncertain data. *This thesis demonstrates that it is possible to effectively manage large, imprecise databases using a generic approach based on probability theory.*

One example application domain for a probabilistic database, such as our system MystiQ, is integrating data from autonomous sources [21]. In this application, a key problem is that the same entity may be represented differently in different sources. In one source, the Very Large Database Conference may be represented by its full name; in another source, it is referenced as simply, "VLDB". As a result, if we ask a query such as, *"Which papers appeared in the last VLDB?"*, a standard database will likely omit some or all of the relevant answers. When integrating large data sources, it is infeasible to manually reconcile every pair of references, so automatic techniques are used that produce scored candidate matches for each pair of references. In MystiQ, we model these scores as *probabilistic events*. In response to a (potentially complicated) SQL query, MystiQ combines the stored probabilistic events to produce a set of answers that are *annotated with the probability that the answer is in the output of the query*. An example output of MystiQ is shown in Figure 1.1. MystiQ makes essential use of the scores associated with answers in two ways: First, by allowing the system to return some answers with lower confidence, we can achieve a *higher recall*; that is, we are able to return more of the papers that truly appeared in VLDB. Second, MystiQ uses the probability scores to rank the answers which allows *high precision*, that is, spurious papers are ranked below papers that actually appeared in VLDB. The true power of MystiQ is evident in more sophisticated applications. For example, if we perform a social networking analysis to find

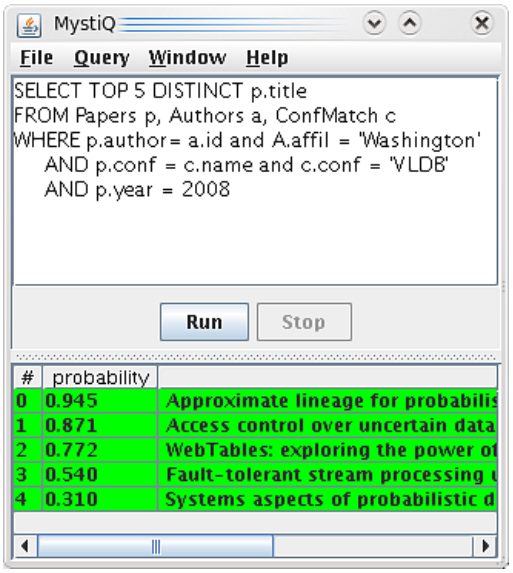


Figure 1.1: A screen shot of MystiQ [21, 142], which ranks query answers by probability; the probability is computed by combining the uncertainty in the database tables mentioned in the query.

influential papers, we could incorporate the (imprecise) results into our paper database. We could then pose interesting structured queries, such as, *"find influential papers from previous VLDBs that were authored at the University of Washington"*.

1.1 Problem Space, Key Challenges, and Goals

The key technical goal of this dissertation is to build a probabilistic relational database system that is able to scale to large databases (gigabytes or more) with query performance that is comparable to a standard relational database without imprecision. We believe that we have succeeded in this goal as the techniques in this dissertation allow sophisticated SQL queries to be run on tens of gigabytes of relational data using the MystiQ system, which demonstrates our central thesis: it is possible to effectively manage large, imprecise databases using a generic approach based on probability theory. We briefly discuss the problem space of this dissertation (a more substantial discussion of related work is in Chapter 7).

Datamodel

In this work, we focus exclusively on discrete relational databases that are queried using SQL or its formal equivalent, conjunctive queries [1]. This is not the only choice: Others have chosen in the last few years to consider probabilistic XML databases [3, 36, 104, 150], streaming probabilistic databases queried using sequence languages [99, 138, 157], and continuous probabilistic, sensor databases [34, 56]. Even within the space of discrete probabilistic relational databases, there are a wide-variety of different approaches: The Monte Carlo DB or MCDB [95] is an approach that is well-grounded in statistics; it allows one to specify complex distributions over attributes and even entire tables. The distributions are specified implicitly, system can handle any distribution that a user is able to sample from. Processing these black-box distribution functions efficiently is very difficult, and the key challenge the project addresses is performance. Although this model seems substantially different from the approach we consider in this dissertation, we believe that many of our techniques are applicable: As noted by Jampani *et al.* [95] in the paper that introduces MCDB, the techniques of Chapter 5 dealing with materialized views are a promising technique to enable scalability. Another line of work casts the problem of probabilistic query answering as probabilistic inference [148, 167]. This approach allows these systems to leverage years of statistical inference expertise in coping with complex distributions. In contrast, the approach in this dissertation starts with a simpler model, and attempts to leverage data management techniques for scalability such as materialized views (Chapter 5) , and database theory techniques to find the border of tractability (Chapter 4).

Two projects are very closely related to the MystiQ project and the contents of this dissertation: the Trio system [17, 129, 168] and the MayBMS system [8, 9, 91]. The original focus of the Trio was understanding the modeling issues that arise when keeping imprecise or uncertain information in the database. Notably, they promoted the idea of *lineage* which helps process probabilistic queries; a concept we discuss in detail in Chapter 2. In contrast, the focus of this dissertation is on performance. The MayBMS system supports a more powerful query language than standard SQL that allows one to do interesting operations such as introducing uncertainty via the query or conditioning on complex probabilistic events. In this dissertation, we will consider only SQL or conjunctive queries. One benefit of choosing a simple model is that many of the results in this dissertation are

applicable to the systems that we have discussed above.

Key Technical Challenges and Goals

The key challenge that we address in this dissertation is performance. Performance is challenging in probabilistic databases: in contrast to standard SQL processing, which is always theoretically easy (AC^0), evaluating queries on a probabilistic database is theoretically hard ($\sharp\mathsf{P}$-hard) [46, 78][1]. While hardness is a key challenge, it is also a golden opportunity: Optimization techniques from the database management literature can actually become more effective for probabilistic databases than they were for standard relational databases. For example, materialized views are an effective technique in relational optimizers, which allows the system to precompute information and then use this precomputed information to optimize queries. In probabilistic databases, this technique is even more effective: As we will see in Chapter 5, a query may be theoretically hard, but by precomputing information, at run-time the remaining processing can be done efficiently (in PTIME). Practically, using materialized views we are able to reduce the cost of query processing on probabilistic databases substantially (from hours to seconds).

1.2 Technical Contributions

We divide the techniques of this dissertation into two categories: *query-time techniques*, which are efficient run-time query-processing strategies, and *view-based techniques*, which are techniques that exploit logical views and precomputation to optimize queries.

Query-time techniques

Top-K Processing Computing the output score of even a single answer on a probabilistic database requires techniques that are orders of magnitude slower than processing standard SQL, such as Monte Carlo sampling. Often, users are not interested in all answers, but only the most highly

[1]Here, our notion of complexity is data complexity in which the query is fixed, but the database is allowed to grow. We measure the complexity with respect to this growth. AC_0 is the class of languages that can be decided by uniform families of circuits with unbounded fan-in, polynomial size and constant depth [128, pg. 386]. $\sharp\mathsf{P}$ is a class of counting problems: For any decision problem that can be verified in P, a $\sharp\mathsf{P}$ problem asks for the number of solutions of that problem. The canonical $\sharp\mathsf{P}$ hard problem is $\sharp$SAT, which counts the number of solutions of a Boolean formula [128, pg. 441].

ranked, say top 10, answers. In spite of this, they are forced to wait as the system churns through thousands of useless results. To remedy this, we designed an algorithm called *multisimulation* that saves computational resources in two ways: (1) it focuses on those answers that have a chance of being in the top 10, and (2) instead of computing a precise probability for each answer, it computes only a relative ordering of the top answers. The central technical result of this chapter the multisimulation algorithm is optimal among all deterministic algorithms for finding the top k most highly probable tuples. Moreover, it is within a factor of 2 of *any* (even non-deterministic) approach. Multisimulation is the heart of MystiQ's processing of general SQL queries and is the subject of Chapter 3. This work appeared in the International Conference on Data Engineering in 2007 [137] and is joint work with Nilesh Dalvi and Dan Suciu.

Probabilistic Decision Support The next generation of business applications will generate large quantities of imprecise data, and so will be difficult for users to understand. Traditional decision support systems allow users to understand their (precise) databases by providing sophisticated aggregation functions, e.g., using the `HAVING` clause in SQL. Inspired by the success of decision support queries in relational databases, we studied the evaluation of decision support queries on probabilistic databases. The key challenge is that standard approaches to processing imprecise queries, such as sampling, force the user to choose between poor quality guarantees or prohibitively long execution times. To overcome this, we designed an algorithm to efficiently and exactly evaluate decision support queries using generalized arithmetic operations[2]. For a class of decision support queries, our algorithms are optimal: (1) either a query can be computed efficiently using our generalized arithmetic operations and is in PTIME, or (2) computing an aggregation query's exactly probability is intractable (♯P-hard), but our results can provide a provably efficient approximation (an FPTRAS), or (3) it is both intractable and does not have an FPTRAS. Our results provide a guide to extend a probabilistic database to process decision support queries and define the limits of any approach. This work is the subject of Chapter 4 and is joint work with Dan Suciu. A preliminary version of this work appeared in the Symposium on Databases and Programming Languages, 2007 [139], and

[2]Formally, we used an abstraction based on semirings and monoid convolutions. Monoids are an abstraction of addition (or multiplication) on the natural numbers. Semirings can be viewed as an abstraction of the standard rules of multiplication and addition applied to Monoids.

a extended version appeared in the Journal of Very Large Databases [143].

View-based techniques

Materialized Views for Probabilistic Databases In standard databases, a fundamental query optimization technique is to precompute intermediate results, known as *materialized views*, and then use these views to expedite future queries. If we directly apply this technique to probabilistic databases, then we run into a problem: each intermediate result in the view (and answer to a query) is actually a probabilistic event and so may be *correlated with other intermediate results*. The standard approach to tracking correlations requires expensive bookkeeping, called *lineage* [168], that records every derivation of every answer that appears in the view. To recover the classical performance benefits of materialized views, we need to discard the lineage, but doing so may lead to incorrect results. Our key technical result is a simple and efficient test to decide whether one can safely discard the lineage associated with a view. On rare occasions, this test may return a false negative, meaning that it rejects some views that could be safely processed. Practically, one cannot go too far beyond this test, because we showed that any complete test (without false positives) must be inefficient: the problem is Π_2P-Complete in the size of the view definition[3]. We validated our approach using data from a Seattle-area start-up company, iLike.com, and showed that probabilistic databases are able to handle huge amounts (GBs) of imprecise data. This work is the subject of Chapter 5. It was joint work with Dan Suciu and appeares in the Very Large Database Conference of 2007 [140] and will appear in the Journal of Computer and System Sciences [49].

Approximate Lineage In addition to performance challenges, correlations make it difficult to understand and debug the output of a probabilistic database. Specifically, the sheer size and complexity of the correlation information (lineage) can overwhelm both the query processor and the user of the system. As the amount of lineage increases, the importance of any individual piece of lineage decreases. Inspired by this observation, we propose a scheme called *sufficient lineage* that removes some correlation information to produce a smaller *approximate lineage formula*. With a substantially smaller lineage formula (often hundreds of times smaller), a probabilistic database is able to

[3] Π_2P denotes the second level of the polynomial hierarchy, which is the class of problems decidable by a coNP machine with access to an NP oracle [128, pg. 425].The canonical complete problem for Π_2P is $\forall\exists$SAT.

process many queries orders of magnitude more efficiently than with the original, full lineage formula. Although a sufficient lineage formula is much smaller than a traditional lineage formula, we demonstrated theoretically and empirically that a probabilistic database using sufficient lineage still provides high quality answers. Further, sufficient lineage has two practically important properties: first, it is syntactically identical to standard lineage, and so can be used without modifying the query processor, and second, it encodes a small set of explanations for a query answer, and so can be used to debug query answers. Sufficient lineage is only one instantiation of approximate lineage: We also defined other formalisms for approximate lineage, notably, one based on polynomials. This formalism can allow smaller lineage functions, but the formulae it produces are *not* syntactically identical to standard lineage. Thus, polynomial lineage requires additional work to integrate with a probabilistic database. This was joint work with Dan Suciu and appeared in the proceedings of the Very Large Database Conference in 2008 [141].

Summary The central goal of this dissertation is to provide a framework that can manage large amounts of probabilistic data, which supports the thesis of this work: it is possible to effectively manage large, imprecise databases using a generic approach based on probability theory. The main technical challenge is performance, and the techniques of this dissertation use a two-pronged approach to this problem: *query-time* optimizations and *view-based* or *precompuation* optimizations.

Chapter 2

PRELIMINARIES

This section gives two critical pieces of background material for this dissertation: how to represent (store) a probabilistic database and how to query it. We describe each of these two pieces in two different ways: first, we describe a concrete representation of a probabilistic database called *Block-independent-disjoint* or (BID) [137, 140], and then a more abstract representation system based on *lineage* [168] and *c*-tables [93]. As we show here, if we add conjunctive views to BID tables then the two representations essentially equivalent.

The strengths of the two approaches are complementary. We introduce the BID representation because it is *general* and *concrete*. It is general since it includes as special cases many other representations discussed in the literature such as *p*-?-sets and *p*-or-sets [82], ?- and *x*-relations [145], and tuple independent databases [46, 108]. The BID representation is *concrete* in that it is the representation implemented in the Mystiq system [21]. On the other hand, the representation based on lineage allows us to specify the technical contributions later in this work precisely and succinctly. The formal model that underlies both representations is the same: *the possible worlds model* [64]. This model is the standard for probabilistic databases and is essentially a reformulation of standard discrete probability theory.

2.1 General Database Preliminaries

We briefly recall some standard database concepts to fix our notation[1].

2.1.1 A Declarative View of Queries

Fix a finite or infinite domain $\mathbb{D}$, which will contain all values that occur in our database, e.g., movie names, review ids, etc. in a movie reviews database. We assume a standard relational schema R

[1]Our presentation in this section borrows from the thesis of Gerome Miklau [118] and the text book by Abiteboul, Hull, and Vianu [1]

with relation names $\mathtt{R}_1, \mathtt{R}_2, \ldots$ and follow Datalog notation. For example, $R_1(a, b, c)$ denotes a tuple (a, b, c) in $\mathtt{R}_1$. A *database instance* or *world* J is a subset of tuples in each of the relations. The content (or extent) of a relation $\mathtt{R}$ in a world J is denoted $\mathtt{R}^J$. For a fixed schema, we denote the set of all possible instances with *Inst*.

A *query* of arity k is a function $q : \mathit{Inst} \to \mathcal{P}(\mathbb{D}^k)$, i.e., a query takes as input a database instance and produces a set of tuples with the same schema[2]. In this work, we focus on a subset of queries called *conjunctive queries* [1]. An abstract example of a conjunctive query is

$$q_1(z) :\!- \mathtt{R}(x, z, \text{`c'}), \mathtt{S}(x, y, -), \mathtt{T}(y, -, \text{`a'})$$

Here, x, y, z are variables, 'a' and 'c' are constants, $-$ represents anonymous variables (each $-$ is distinct from all other variables). Denote by $\mathbf{var}(q)$ the set of all variables in a query q and by $\mathbf{const}(q)$ the set of all constants in a query q. In general, a conjunctive query is a rule of the form:

$$q(\boldsymbol{y}) :\!- g_1, \ldots, g_n$$

where each of the g_i are subgoals that may contain selections. The query q_1 above consists of three subgoals. The first subgoal $\mathtt{R}(x, z, \text{`b'})$ contains a selection that intuitively says we are only interested in $\mathtt{R}$ tuples whose third component is 'b'. We denote the set of subgoals in the query q as $\mathbf{goal}(q)$.

We evaluate a query q on an instance J in the standard way: we search for a query homomorphism $h : \mathbf{var}(q) \cup \mathbf{const}(q) \to \mathbb{D}$ which is identity on constants, i.e., $h(c) = c$ for all $c \in \mathbf{const}(q)$ and such that for each subgoal $g_i = \mathtt{R}(z_1, \ldots, z_m)$ where z_i is either a variable or a constant, we have that $\mathtt{R}(h(z_1), h(z_2), \ldots, h(z_m)) \in \mathtt{R}^J$. If this holds for all subgoals, then we return the tuple $(h(y_1), h(y_2), \ldots, h(y_m))$.

Relationship to SQL

One reason that this fragment is important is that it captures the heart of the standard query language for relational databases, SQL [94]. Figure 2.1(b) shows an SQL query that is equivalent to q_1. Later, in Chapter 4, we enhance this language to handle some advanced features of SQL, notably

[2] A (Boolean) query q is a function where $k = 0$ (or is a sentence in the standard sense of first-order logic [61]).

R(A,B,C)
S(A,B,D)
T(B,E,F)

(a)

```
SELECT DISTINCT R.B
FROM R,S,T
WHERE R.A = S.A AND R.C = 'c'
      S.B = T.B AND T.F = 'a'
```

(b)

$$P_1 = \mathrm{R}(x, z, \text{'c'}) \bowtie \pi_{-u_1} \mathrm{S}(x, y, u_1)$$
$$P = \pi_{-y} (P_1 \bowtie \pi_{-u_2} \mathrm{T}(y, u_2, \text{'a'}))$$

(c)

Figure 2.1: (a) The corresponding Relational Style Schema for query q_1. (b) is an SQL query that is equivalent to q_1. (c) P is an equivalent relational query plan (we have labeled the anonymous variables in q as u_1 and u_2 for readability). We use P_1 to aid readability; we simply inline the definition of P_1 in P.

aggregation functions that include functions such as `SUM` or `AVG`.

2.1.2 An operational view of conjunctive queries

Query plans are a more operational, but equivalent, way to describe conjunctive queries. A query plan is tree of relational operators, where each relational operator is a function from relations (of any arity) to relations[3].

Definition 2.1.1 (Query Plan Syntax)**.** *A query plan P is inductively defined as (1) a single subgoal which may contain selections, (2) $P = \pi_{-x}P_1$ (projection) where P_1 is a plan and x is a variable, (3) $P_1 \bowtie P_2$ where P_1 and P_2 are plans.*

Figure 2.1(c) shows a query plan that is equivalent to q_1 (and so to the SQL query in Figure 2.1(b)). We view query plans as inductively computing tuples. We also denote the variables that are "returned" by a plan P with $\mathbf{var}(P)$ and define it inductively: If $P = g$ then $\mathbf{var}(P)$ is the set of free variables in g, if $P = \pi_{-x}P_1$ then $\mathbf{var}(P) = \mathbf{var}(P_1) - \{x\}$, and if $P = P_1 \bowtie P_2$ then $\mathbf{var}(P) = \mathbf{var}(P_1) \cup \mathbf{var}(P_2)$.

[3]Our description of relational query plans is standard with the minor caveat that we view projections as removing attributes, rather than naming which attributes they keep.

Query Plan Semantics

A query plan P on an instance I produces a set of tuples. To define this set, we first recall *term matching* [159]: A tuple t matches a subgoal $g = R(y_1, \ldots, y_n)$ where the y_i may be (repeated) variables or constants, if there exists a homomorphism $h : \mathbf{var}(g) \cup \mathbf{const}(g) \to \mathbb{D}$ such that: $t = R(h(y_1), \ldots, h(y_n))$. For example, if $g = R(x, x, y, \text{'a'})$ then the tuple $R(b, b, c, a)$ matches g, but neither of the following tuples match g: $R(a, b, c, a)$ (there is no way to map x) nor $R(a, a, b, b)$ (the constants cannot match). Now, we can define the semantics of query plans:

We write $t \in P(I)$ if the tuple t is produced by P on instance I and define this set of tuples inductively:

- Let $P = g$ with $g = R(\boldsymbol{y})$, a subgoal with selections, then $t \in P$ if $t \in R^I$ matches g.
- Let $P = \pi_{-x} P'$ then $t \in P$ if there exists t' such that $t'[\mathbf{var}(P)] = t[\mathbf{var}(P)]$ and $t' \in P'$.
- Let $P = P_1 \bowtie P_2$ then $t \in P$, if there exists t_1, t_2 such that $t_i[\mathbf{var}(P_i)] = t[\mathbf{var}(P_i)]$ for $i = 1, 2$.

As is standard, the semantics of query plans are equivalent to the semantics of conjunctive queries. Figure 2.1(c) illustrates a query plan that is equivalent to the conjunctive query in Figure 2.1(a) (and so to the SQL query in Figure 2.1(b)).

2.2 BID databases

The representation of a probabilistic database that is used in the Mystiq system is called Block-Independent Disjoint or BID. A BID database consists of one or more tables, where each table is associated with a BID schema. A BID schema describes both the content of the table as in standard relational databases, and additionally, the schema describes how we should interpret the imprecision in that table. An example BID database about movie reviews is shown in Figure 2.2.

More precisely, a BID schema is a relational schema where the attributes are partitioned into three disjoint sets. We write a BID schema for a symbol R as $\mathtt{R}(\boldsymbol{K}; \boldsymbol{A}; P)$ where the sets are separated by semicolons. Here, $\boldsymbol{K}$ is a set of attributes called the *possible worlds key*; $\boldsymbol{A}$ is a set of attributes called the set of *value attributes*; and P is a single, distinguished attribute that stores the marginal probability of the tuple called the *probability attribute*. The values of $\boldsymbol{K}$ and $\boldsymbol{A}$ come from some discrete domain, and the values in the attribute P are numbers in the interval $(0, 1]$. In the relation, $\boldsymbol{K}$ and $\boldsymbol{A}$ must form a key, i.e., if $t[KA] = t'[KA]$ then $t = t'$. For example, one BID schema

ReviewID	Reviewer	Title	P	
231	‘Ryan’	‘Fletch’	0.7	t_{231a}
		‘Spies Like Us’	0.3	t_{231b}
232	‘Ryan’	‘European Vacation’	0.90	t_{232a}
		‘Fletch 2’	0.05	t_{232b}
235	‘Ben’	‘Fletch’	0.8	t_{235a}
		‘Wild Child’	0.2	t_{235b}

`Reviews`(ReviewID, Reviewer ; ReviewTitle ; P)

Title	Matched	P	
‘Fletch’	‘Fletch’	0.95	m_1
‘Fletch’	‘Fletch 2’	0.9	m_2
‘Fletch 2’	‘Fletch’	0.4	m_3
‘Golden Child’	‘Golden Child’	0.95	m_4
‘Golden Child’	‘Golden Child’	0.8	m_5
‘Golden Child’	‘Wild Child’	0.2	m_6

`MovieMatch`(CleanTitle, ReviewTitle ;; P)

Figure 2.2: Sample data arising from integrating automatically extracted reviews from a movie database. `MovieMatch` is a probabilistic relation, we are uncertain which review title matches with which movie in our clean database. `Reviews` is uncertain because it is the result of *information extraction*. Note that the identifiers next to each tuple are not part of the database, e.g., t_{232}, they are there so that we can refer to tuples by a shorthand.

in Figure 2.2 is `Reviews`(ReviewerID,Reviewer;Title;P): {ReviewerID,Reviewer} is the possible worlds key, {Title} is the set of value attributes, and P is the probability attribute. Also pictured in Figure 2.2 is an instance of this schema.

Semantics of BID tables

An instance of a BID schema represents a distribution over instances called *possible worlds* [64]. The schema of a possible worlds is $R(\boldsymbol{K}, \boldsymbol{A})$, i.e., the same schema as the BID table except *without* the attribute P. Again, this models our choice that the user should query the data as if it were deterministic, and it is the goal of the system to manage the imprecision. Let J be an instance of a BID schema. We denote by $t[\boldsymbol{KA}P]$ a tuple in J, emphasizing its three kinds of attributes, and call $t[\boldsymbol{KA}]$, its projection on the $\boldsymbol{KA}$ attributes, a *possible tuple*. Define a *possible world*, W, to be any instance of $R(\boldsymbol{K}, \boldsymbol{A})$ consisting of possible tuples such that $\boldsymbol{K}$ is a key[4] in W. Note that the key constraints do not hold in the BID instance J, but must hold in any possible world W. Let $\mathcal{W}_J$ be the set of all possible worlds associated to J. In Figure 2.2, one possible world W_1 is such that $R^{W_1} = \{t_{231a}, t_{232b}, t_{235a}\}$. Graphically, we have illustrated the blocks from which the BID representation gets its name: Each key value defines a block of tuples such that at most one of them

[4]We say that $\boldsymbol{K}$ is a key in W if for $t, t' \in R^W$ we have $t[\boldsymbol{K}] = t'[\boldsymbol{K}] \implies t = t'$.

may occur together in any possible world. For example, no possible world may contain both t_{232a} and t_{232b}.

We define the semantics of BID instances only for *valid* instances, which are BID instances such that the values in P can be interpreted as a valid probability distribution, i.e., such that for every tuple $t \in R^J$ in any BID relation $\mathtt{R}(\boldsymbol{K}; \boldsymbol{A}; P)$ the inequality $\sum_{s \in \mathtt{R}: s[\boldsymbol{K}]=t[\boldsymbol{K}]} s[P] \leq 1$ holds. A valid instance J defines a finite probability space $(\mathcal{W}_J, \mu_J)$. First note that any possible tuple $t[\boldsymbol{KA}]$ can be viewed as an event in the probability space $(\mathcal{W}_J, \mu_J)$, namely the event that a world contains $t[\boldsymbol{KA}]$. Then we define the semantics of J to be the probability space $(\mathcal{W}_J, \mu_J)$ such that (a) the marginal probability of any possible tuple $t[\boldsymbol{KA}]$ is $t[P]$, (b) any two tuples from the same relation $t[\boldsymbol{KA}]$, $t'[\boldsymbol{KA}]$ such that $t[\boldsymbol{K}] = t'[\boldsymbol{K}]$ are *disjoint events* (or *exclusive events*), and (c) for any set of tuples $\{t_1, \ldots, t_n\}$ such that all tuples from the same relation have distinct keys, the events defined by these tuples are independent. If the database contained only the `Reviews` table, and $W_1 = \{t_{231a}, t_{232b}, t_{235a}\}$, we see that $\mu(W_1) = .7 * .9 * .8 = .504$.

Example 2.2.1 The data in Figure 2.2 shows an example of a BID database that stores data from integrating extracted movie reviews from USENET with a movie database from IMDB. The `MovieMatch` table is uncertain because it is the result of an automatic matching procedure (or fuzzy-join [31]). For example, the probability a review title 'Fletch' matches a movie titled 'Fletch' is very high (0.95), but it is not certain (1.0) because the title is extracted from text and so may contain errors. For example, from *'The second Fletch movie'*, our extractor will likely extract just *'Fletch'* although this review actually refers to *'Fletch 2'*. The review table is uncertain because it is the result of *information extraction*. That is, we have extracted the title from text (e.g.*'Fletch is a great movie, just like Spies Like Us'*). Notice that $t_{232a}[P] + t_{232b}[P] = 0.95 < 1$, which indicates that there is some probability reviewid 232 is actually not a review at all.

Remark 2.2.2. *Recall that two distinct possible tuples, say $t[KA]$ and $t'[KA]$, are disjoint if $t[K] = t'[K]$ and $t[A] \neq t'[A]$. But what happens if $A = \emptyset$, i.e., all attributes are part of the possible worlds key? In that case all possible tuples become independent, and we sometime call a table $R(K;;P)$ a* tuple independent table *[46], which is also known as a* ?-table *[129] or a p-*?-table *[82]. An example is the* `MovieMatch` *table in Figure 2.2.*

Finally, we generalize to probabilistic databases that contain many BID tables in the standard way: tuples in distinct tables are independent.

2.2.1 Probabilistic Query Semantics

Users write queries in terms of the possible worlds schema. In particular, a query written by the user does *not* explicitly mention the probability attributes of the BID relations. This reflects a choice that the user should pose her query as if the data is deterministic, and it is the responsibility of the system to combine the various sources of imprecision. Other choices are possible and useful for applications. For example, the work of Koch [106], Moore *et al.* [120], and Fagin *et al.* [64] discuss query languages that allow predication on probabilities, e.g. is the probability of a query q greater than 0.5?; Moreover, the works of Koch and Fagin *et al.* allow these predicates to be used compositionally within queries. In this dissertation, the answer to any query is a set of tuples annotated with probability scores. The score is exactly the marginal probability that the query is true, and we define this formally for any Boolean property:

Definition 2.2.3 (Query Semantics)**.** *Let* $\Phi : Inst \to \{0, 1\}$ *be a Boolean property, then the marginal probability of any* Φ *on a probabilistic database* $\mathcal{W} = (J, \mu)$ *is written* $\mu_J(\Phi)$ *is defined by:*

$$\mu_J(\Phi) = \sum_{W \in \mathcal{W}_J : W \models \Phi} \mu_J(W)$$

where we write $W \models \Phi$ *if* $\Phi(W) = 1$.

A Boolean conjunctive query q is a Boolean property and so, we write $\mu_J(q)$ to denote the marginal probability that q is true.

Example 2.2.4 For example, the query that asks *"was the movie 'Fletch' reviewed?"* or as a Boolean conjunctive query, $q() = \mathtt{R}(-, -, \text{'Fletch'})$, is true when t_{231a} or t_{235a} are present. Definition 2.2.3 tells us that, semantically, we can compute the probability that this occurs by summing over all possible worlds. Of course, there is a simple *ad hoc* shortcut in this case: $\mu(q) = 1 - (1 - 0.7)(1 - 0.8) = 0.94$.

For non-Boolean queries, we (semantically) substitute the head variables with constants in all possible ways. The result of this process is a Boolean query. For example, $q(x) :- \mathtt{R}(x, -, \text{'Fletch'})$

returns all people who may have reviewed the movie ‘Fletch’. The score of an answer, say that Ryan reviewed the movie, is given by the Boolean query $q'() :-$ (‘Ryan’, –, ‘Fletch’). The probability of q' is precisely the scores associated with the answer Ryan that is returned to the user in the Mystiq system (as shown in Figure 1.1).

2.3 Lineage, c-tables, or Intensional evaluation

In a probabilistic database, the presence or absence of any tuple is a probabilistic event. Thus, a Boolean query – whose value may depend on many tuples in the database – defines a complex event which may contain correlations between tuples. *Lineage* is a mechanism that tracks these correlations by essentially recording all derivations of a query [168]; this is similar to our informal reasoning in Example 2.2.4.

In this dissertation, we adopt a viewpoint of lineage based on *c*-tables [82, 93]: we think of lineage as a constraint that tells us which worlds are possible. This viewpoint still results in the standard *possible worlds semantics* for probabilistic databases that we saw in the previous section. We introduce this abstraction because it allows us to unify the presentation of the technical contributions in this work.

2.3.1 A declarative view of Lineage

The starting point for lineage is that there is a set of *atoms* (or *base events*) denoted $\mathcal{X} = x_1, x_2, \ldots$ which are propositions about the real world, e.g., a Boolean event could encode that “Ryan reviewed Fletch”. Atoms may also be non-Boolean and then can be thought of as variables that take a value in some domain $\mathbb{A}$. An *EQ-term* is a formula $x = a$ where $x \in \mathcal{X}$ and a is an element of $\mathbb{A}$. Informally, an EQ-term is true if x takes the value a and false otherwise. A lineage function, λ, assigns to each tuple t a Boolean expression over EQ-terms which is denoted $\lambda(t)$. These Boolean expressions are precisely the conditions from which the *c*-tables get their name [93]. When all atoms are Boolean, we abbreviate $x = true$ as simply x for readability.

Example 2.3.1 Figure 2.3 shows the `Reviews` table from the previous example (Example 2.2) encoded in the syntax of lineage. Here, if the atom t_{232} takes the value a then this implies that in review 232 Ryan reviewed ‘European Vacation’ while if t_{232} takes value b this means that he instead

ReviewID	Reviewer	Title	λ
231	'Ryan'	'Fletch'	$x_{231} = a$
		'Spies Like Us'	$x_{231} = b$
232	'Ryan'	'European Vacation'	$x_{232} = a$
		'Fletch 2'	$x_{232} = b$
235	'Ben'	'Fletch'	$x_{235} = a$
		'Wild Child'	$x_{235} = b$

`Reviews`(ReviewID, Reviewer, ReviewTitle, λ)

x_{231}	a	0.7
x_{231}	b	0.3
x_{232}	a	0.9
x_{232}	b	0.05
x_{232}	c	0.05
x_{232}	a	0.8
x_{232}	b	0.2

A Probability Assignment

Figure 2.3: The `Reviews` relation encoded in the syntax of Lineage.

reviewed 'Fletch 2'. Since we insist that t_{232} has a single value, he could not have reviewed both. He may, however, have reviewed neither, if for example, $t_{232} = c$.

Semantics of Lineage

To define the standard semantics of lineage, we define a *possible world* W through a two-stage process: First, select an assignment A for each atom in $\mathcal{X}$, i.e., $A : \mathcal{X} \to \mathbb{A}$. Then, for each tuple t, include t if and only if $\lambda_t(A)$ evaluates to true. Here, $\lambda_t(A)$ denotes the formula that results from substituting each atom $x \in \mathcal{X}$ with $A(x)$. This process results in an unique world W for any assignment A. We denote the set of all assignments to $\mathcal{X}$ as Assign($\mathcal{X}$) or simply Assign.

Example 2.3.2 One assignment is $x_{231} = a$, $x_{232} = c$ and $x_{235} = a$. Then the resulting possible world W contains two tuples $W_1 = \{(231, \text{Ryan}, \text{Fletch}), (235, \text{Ben}, \text{Fletch})\}$. In contrast, the world $W_2 = \{(231, \text{Ryan}, \text{Fletch}), (231, \text{Ryan}, \text{Spies Like us})\}$ is *not* possible as this would require that $x_{231} = a$ and $x_{231} = b$.

We capture this example in a definition. Fix a relational schema. A *world* is a subset of tuples conforming to that relational schema. Given an assignment A and a world W, we say that W is the *possible world* induced by A if it contains exactly those tuples consistent with the lineage function, that is, for all tuples t, $\lambda_t(A) \iff t \in W$. Moreover, we write $\lambda(A, W)$ to denote the Boolean function that is true when W is a possible world induced by A. In symbols,

$$\lambda(A, W) \stackrel{\text{def}}{=} \left(\bigwedge_{t : t \in W} \lambda_t(A) \right) \wedge \left(\bigwedge_{t : t \notin W} \neg \lambda_t(A) \right) \tag{2.1}$$

We complete the construction of a probabilistic database as a distribution over possible worlds by introducing a score function. We assume that there is a function p that assigns to each atom $x \in \mathcal{X}$ and each atom $a \in A$ a probability score denoted $p(x = a)$. In Figure 2.3 $p(x_{232} = a)$ has been assigned a score indicating that we are very confident that Ryan reviewed European Vacation in reviewid 232 (0.9). An important special case is when $p(x = a) = 1$, which indicates absolute certainty.

Definition 2.3.3. *Fix a set of atoms $\mathcal{X}$ with domain $\mathbb{A}$. A* probabilistic assignment *p is a function that assigns a probability score to each atom value pair $(x, a) \in \mathcal{A} \times \mathbb{A}$ which reflects the probability that $x = a$, we write $p(x, a)$ as $p(x = a)$ for readability. Thus, for each $x \in \mathcal{A}$ we have $\sum_{a \in \mathbb{A}} p(x = a) = 1$.*

A probabilistic lineage database *$\mathcal{W}$ is a probabilistic assignment p and a lineage function λ that represents a distribution μ over worlds defined as:*

$$\mu(W) \stackrel{\text{def}}{=} \sum_{A \in \text{Assign}} \lambda(A, W) \prod_{x,v:A(x)=v} p(x = v)$$

In words, $\mathcal{W}$ is a product space over $\mathbb{A}^{\mathcal{X}}$ with measure p.

Since for any A, there is a unique W such that $\lambda(A, W) = 1$, μ is a probability measure. We then define the semantics of queries exactly as in Definition 2.2.3.

Example 2.3.4 Consider a simple query on the database in Figure 2.3:

$$q() = \mathtt{R}(-, -, \text{‘Fletch’}), \mathtt{R}(-, -, \text{‘Fletch 2’})$$

This query asks if the movies ‘Fletch’ and ‘Fletch 2’ were both reviewed in our database. This query is true in a world when the following (lineage) formula holds $(x_{235} = a \vee x_{231} = a) \wedge x_{231} = a$. Said another way, if we think of q as a view, then its output is a single tuple with $\lambda(q()) = (x_{235} = a \vee x_{231} = a) \wedge x_{231} = a$.

Generalizing this example, one may wonder whether we can always find a formula (or constraint) that correctly captures the semantics of the derived table. The answer is yes, which is exactly the strong representation property of Imielinksi and Lipski for c-tables [93, Theorem 7.1].

2.3.2 An operational view of Lineage

In databases, new tuples are derived from old tuples in the database using queries (views). An important case is when the queries that define the views are conjunctive, as in Example 2.3.4. Recall that a conjunctive query can be computed using a relational plan P. This plan is useful to us here because we can use it to compute the lineage derived by the query in a straightforward way[5]:

Inductively, we compute the lineage of a tuple t using a plan P, $\lambda(P, t)$.

- Let $P = g$, then $\lambda(P, t) = \lambda(t)$ if t matches g otherwise $\perp$ (false)
- Let $P = \pi_{-x} P'$ then $\lambda(P, t) = \bigvee_{t':t'[\mathbf{var}(P)]=t[\mathbf{var}(P)]} \lambda(P', t')$
- Let $P = P_1 \bowtie P_2$ then $\lambda(P, t) = \lambda(P_1, t_1) \wedge \lambda(P_2, t_2)$ where $t_i[\mathbf{var}(P_i)] = t[\mathbf{var}(P)]$

It is easy to check using any plan that is logically equivalent to q in Example 2.3.4 that we can compute the same lineage operationally using the above rules, as we computed in an *ad hoc* way in the example.

Later, we generalize this technique of computing lineage in two ways: To compute aggregate queries, we follow Green *et al.* [81] and generalize lineage to compute formulas in (non-necessarily-Boolean) semirings. In an orthogonal direction, we reduce the expense of processing and understanding lineage, by approximating the Boolean formulas during lineage computation, a technique called *approximate lineage*.

Internal Lineage An important special case of lineage is called *internal lineage* [145]; the intuition is that internal lineage is built-up exclusively from views (or queries) inside the database. In contrast, *external lineage* models the situation when lineage is also created by external entities, e.g., during an Extract-Transform-Load (ETL) workflow. More precisely, a relation R is a *base relation* if for each tuple t in R $\lambda(t)$ consists of a single EQ-term. An internal lineage database contains base relations and views (whose lineage functions are built using the previous rules). It is interesting to note that BID tables are essentially base relations with an additional schema-like restriction: there is a possible worlds key K that satisfies the following two conditions: Let $t, t' \in R$

1. If $t[K] = t'[K]$ then $\mathbf{var}(\lambda(t)) = \mathbf{var}(\lambda(t'))$ (Key values are variables)
2. if $\lambda(t) = \lambda(t')$ then $t = t'$ (λ is injective)

[5]This computation is similar to the Probabilistic Relational Algebra of Fuhr and Roelleke [68].

Thus, BID tables are a special case of lineage, but under very mild restrictions.

2.3.3 The role of lineage in query evaluation

A simple, but important, property of conjunctive queries (views) on BID tables is that the lineage formulas they produce can be written as k-DNF formulas[6] for small values of k, i.e., k depends only the size of the query q, but *does* not depend on the size of the data. More precisely,

Proposition 2.3.5. *Let λ be the derived lineage for a tuple t in the output of a conjunctive query q with k subgoals. Then, λ can be written as a k-DNF formula of polynomial size.*

Proof. Using the standard equivalence of relational plans, there is always relational query plan P in which the projections induce a single connected component in the plan tree that contains the root (i.e., the projections are "on top"). Since a projection takes only a single input, this connected component is necessarily a chain. Then, the lineage for each tuple produced by the subplan P_1 of this chain can be seen to be a conjunction of atomic terms: the only operator that modifies the lineage is the join ($\bowtie$), and each join introduces at most one $\wedge$ symbol per tuple. Thus, the number of conjunctions is upper bounded by the number of joined tables, which is k. Since the chain of projections creates a single disjunction, the output formula is a k-DNF. □

The importance of this proposition is that we have reduced our problem of computing the probability a query is true to the problem of computing the probability that a k-DNF formula is satisfied, which is a well-studied problem. Unfortunately, this problem is hard for 2-DNFs[7]. The good news is that this type of formula has efficient approximations a fact that we leverage in Chapter 3. Moreover, in Chapter 4, we generalize this condition to efficiently evaluate and approximate aggregate queries, and in Chapter 6, we exploit this connection to produce small, approximate lineage formula.

---

[6] A k-DNF formula is a Boolean formula in *disjunctive normal form* where each monomial contains at most k literals, i.e., that can be written $\phi = \bigvee_{i=1,m} \bigwedge_{j=1,k} x_{i,j}$ where $x_{i,j}$ is a literal or its negation.

[7] It is easy to see that the query $R(x), S(x,y), R(y)$ can produce *any* 2-DNF formula. Computing the probability that even a monotone a 2-DNF is satisfied is ♯P-hard [162]. This fact was used by Grädel *et al.* [78] to show that computing reliability of queries is ♯P-hard. This result was further sharpened by Dalvi and Suciu [46] to the precise syntactic boundary where hardness occurs.

2.4 Expressiveness of the Model

In this dissertation, we consider a data model where probabilities are listed explicitly. For example, if one has a `Person` table with `salary` and `age` attributes whose values are correlated probability distributions, then in our model one needs to enumerate explicitly all combinations of `salary` and `age`, e.g. `(Smith, 20, 1000, 0.3)`, `(Smith, 20, 5000, 0.1)`, `(Smith, 40, 5000, 0.6)`. This allows for correlated attributes – as long as the joint distribution is represented explicitly. With the addition of views, *any* possible worlds distribution can be represented.

More precisely, we say that a possible worlds distribution $I = (\mathcal{W}_I, \mu_I)$ represents a possible worlds distribution $J = (\mathcal{W}_J, \mu_J)$ if there exists a mapping $H : \mathcal{W}_I \to \mathcal{W}_J$ such that (1) for each relational symbol R in W_J and $W \in \mathcal{W}_J$, $R^W = R^{H(W)}$, i.e., every relation in J has exactly the same content, and (2) for any world $W_J \in \mathcal{W}_J$, we have $\mu(W_J) = \mu(\{W_I \in \mathcal{W}_I \mid H(W_I) = W_J\})$, i.e., the probability is the same[8].

Proposition 2.4.1. *For any possible worlds database $J = (\mathcal{W}, \mu)$, there exists a BID database with conjunctive views whose possible worlds represent J.*

Proof. Let $\mathcal{W} = \{W_1, \ldots, W_n\}$ and introduce new constants $w_1, \ldots, w_n$ for each such world. Let $A(X; W; P)$ be a fresh relational symbol not appearing in J. Then, let $A = \{(x, w_i, \mu(W_i)) \mid W_i \in \mathcal{W}\}$. Notice that since there is only one value for the X attribute, all these events are disjoint: in any possible world the unique variable x takes a single value For each symbol $R(\boldsymbol{A})$ in J, introduce a new symbol $R'(\boldsymbol{A}, W)$, i.e., R' has the same schema as R with an additional attribute W. Now, define the conjunctive view $R(\boldsymbol{x}) :- R'(\boldsymbol{x}, w), A(w)$ where $\boldsymbol{x}$ has the same arity as $\boldsymbol{A}$. For $i = 1, \ldots, n$, define the mapping H by mapping W_i to the unique world where $x = w_i$. We can see that, $R^{W_1} = R^{H(W_1)}$, and so, this distribution represents J. □

If we examine the construction, we see that the database we are constructing is an internal lineage database, since the table A is a base table. We say that a representation system is *complete* if it can represent any discrete possible worlds distribution. Then, we have shown:

Corollary 2.4.2. *BID databases with conjunctive views and internal lineage databases are complete.*

---

[8]This definition necessarily preserves correlations because the choice of H is the same for *all* relations.

This simple result (or a minor variation of it) has been observed in the literature many times [17, 82, 137]. This gives some intuition why variants of this model have been used as far back as 1992 by Barbara *et al.* [13] (without views), and as recently as the current Mystiq [21], Trio [168], Conquer [70], and MayBMS systems [9]: it is a simple, yet surprisingly expressive, model. There are other more succinct models that have been proposed in the literature, which we discuss in Chapter 7.

Chapter 3

QUERY-TIME TECHNIQUE I: TOP-K QUERY EVALUATION

The central observation of this chapter is that on a probabilistic database, the meaningful information is *not* conveyed by the exact scores in a query's answer; instead, the meaningful information is conveyed by *the ranking* of those scores in the query's answer. In many cases, the user is not interested in the scores of every tuple, but likely in only the top few highly ranked answers. With the observation in mind, we focus on efficiently finding and ordering those top tuples, and shift away from prior art that focuses exclusively on computing the exact output probabilities.

In many applications, the motivation for introducing probabilities is to increase the recall of queries on the database. This increase of recall always comes at the expense of lowering precision, which forces the user to deal with a swarm of low-quality answers. Since many tuples in the query answer are of very low quality (probability), and users are often interested in seeing only the most highly ranked answers, it is sufficient to compute just the first k answers for small k. With the idea that a ranking of the top tuples suffices in many situations, we develop a novel query evaluation algorithm for computing the top-k answers that has theoretical guarantees. Additionally, we consider several optimizations that are implemented in the MystiQ system, and we evaluate their impact on MystiQ's performance.

3.1 Motivating Scenario and Problem Definition

We illustrate the challenges that are faced by query evaluation on probabilistic databases using an application that integrates the Internet Movie Database from `imdb.com`, with movie reviews from `amazon.com`. There are over 10M tuples in the integrated database. A simplified schema is shown in Figure 3.1, and we will use it as our running example in this section. Amazon products (DVDs in our case) are identified by a unique Amazon Standard Identification Number, `asin`, and each DVD object has several subobjects: customer reviews, actors, director, etc. The IMDB schema is self explanatory. The value of integrating the two data sources lies in combining the detailed movie data

```
AMZNReviews(asin, title, customer, rating, ...)
AMZNDirector(asin, director)
AMZNActor(asin, actor)
IMDBMovie(mid, movieTitle, genre, did, year)
IMDBDirector(did, dirName)
IMDBCast(mid, aid)
IMDBActor(aid, actorName)
TitleMatch(asin; mid; P)
```

Figure 3.1: Schema fragment of IMDB and Amazon database, and a fuzzy match table

TitleMatch(*asin*; *mid*; *P*)

	asin	mid	**P**
t_1	a282	m897 (“Twelve Monkeys”)	0.4
t_2	(“12 Monkeys”)	m389 (“Twelve Monkeys (1995)”)	0.3
t_3		m656 (“Monk”)	0.013
t_4	a845	m897 (“Twelve Monkeys”)	0.35
t_5	(“Monkey Love”)	m845 (“Love Story”)	0.27

Figure 3.2: Fuzzy matches stored in TitleMatch. The table stores only the asin and mid values, but we included the review title and movie title for readability.

in IMDB with customers ratings in Amazon.

From Imprecisions to Probabilities One source of imprecision in integrating the two sources is that their movie titles often do not match, e.g., Twelve Monkeys v.s. 12 Monkeys or Who Done it? v.s. The Three Stooges: Who Done it. The problem of detecting when two representations denote the same object has been intensively studied, and referred to as deduplication, record linkage, or merge-purge [6, 10, 32, 65, 71, 83, 89, 169, 170]. Perfect object matching is almost always impossible, and when it is possible it is often very costly, since it requires specialized, domain specific algorithms. Our approach is to rely on existing domain independent methods, and change the way their outputs are used. Currently, all fuzzy match methods use a *thresholded similarity function approach* [6], which relies on a threshold value to classify objects into matches and non-matches. This is a compromise that can lead to false positives (when the threshold value is too low) or to false negatives (when the threshold is too high). In contrast, in our approach the system retains all similarity scores and handles them as probabilistic data. We computed a similarity score between each pair of movie title and review title by comparing their sets of 3-grams: this resulted

```
SELECT DISTINCT d.dirName AS Director
FROM AMZNReviews a, AMZNReviews b,
  TitleMatch ax, TitleMatch by,
  IMDBMovie x, IMDBMovie y,
  IMDBDirector d
WHERE a.asin=ax.asin and b.asin=by.asin and x.did=y.did and y.did=d.did
  and x.genre='comedy' and y.genre='drama'
  and abs(x.year - y.year) <= 5 and a.rating>4 and b.rating<2
```

$q(name)$:– Director($name, z$), AMZNReviews(x, r), AMZNReviews(x', r'),
TitleMatch(x, m), TitleMatch(x', m'), IMDBMovie(m, 'Comedy', y, z),
IMDBMovie(m', 'Drama', y', z), $r > 5$, $r' < 2$, abs($z - z'$)

Figure 3.3: Query retrieving all directors that produced both a highly rated comedy and a low rated drama less than five years apart.

in a number p between 0 and 1, which we interpret as the confidence score and stored it in a table called TitleMatch. Figure 3.2 shows a very simplified fragment of TitleMatch(asin;mid;P), consisting of five tuples $t_1, \ldots, t_5$. Each tuple contains an asin value (a review in amazon) and a mid value (a movie in IMDB). The amazon review with asin = a282 refers to a movie with title 12 Monkeys, and this can be one of three movies in the IMDB database: either Twelve Monkeys, or to Twelve Monkeys (1995), or to Monk. Thus, only one of the tuples t_1, t_2, t_3 can be correct, i.e., they are *exclusive*, or *disjoint*, and their probabilities are $p_1 = 0.4$, $p_2 = 0.3$, and $p_3 = 0.013$ respectively. Note that $p_1 + p_2 + p_3 \leq 1$, which is a necessary condition since the three tuples are exclusive events: we normalized the similarity scores to enforce this condition. Similarly, the movie review about Monkey Love can refer to one of two IMDB movies, with probabilities $p_4 = 0.35$ and $p_5 = 0.27$ respectively. We assume that any of the three matches for the first review is independent from any of the two matches of the second review. This summarizes how we mapped fuzzy object matches to probabilities. We briefly discuss other types of imprecisions in Section 3.4.1.

Challenges Query evaluation poses two major challenges. The first is that computing the exact output probabilities is computationally hard. The data complexity for the query in Figure 3.3 is #P-complete (this can be shown using results in [46]), meaning that any algorithm that computes

the probabilities exactly essentially needs to iterate through all possible worlds. Previous work on probabilistic databases avoided this issue in several ways. Barbara *et al.* [13] requires the SQL queries to include all keys in all tables, thus disallowing duplicate elimination. This rules out our query in Figure 3.3, because this query does not include any of the keys of the seven tables in the `FROM` clause. If we included all these keys in the `SELECT` clause, then each director in the answer would be listed multiple times, once for each pair of movies that satisfies the criterion: in our example each of the 1415 directors would have occurred on average 234.8 times. This makes it impossible to rank the directors. ProbView [108] computes probability intervals instead of exact probabilities. In contrast to Luby and Karp's algorithm, which can approximate the probabilities to an arbitrary precision, the precision in ProbView's approach cannot be controlled. In fact, the more complex the query, the wider the approximation intervals, since ProbView's strategies have to conservatively account for a wide range of possible correlations between the input probabilities. For example, when combining the (on average) 234.8 different probabilities to compute the probability of a single director, the resulting interval degenerates to $[0, 1]$ for most directors. One can still use this method in order to rank the outputs, (by ordering them based on their intervals' midpoints) but this results in low precision. Fuhr [68] uses an exponential time algorithm that essentially iterates over all possible worlds that support a given answer. This is again impractical in our setting. Finally Dalvi [46] only considers "safe" queries, while our query is not safe.

The second challenge is that the number of potential answers for which we need to compute the probabilities is large: in our example, there are 1415 such answers. Many of them have very low probability, and exists only because of some highly unlikely matches between movies and reviews. Even if the system spends large amounts of time computing all 1415 probabilities precisely, the users is likely to end up inspecting just the first few of them.

Overview of our Approach

We focus the computation on the top k answers with the highest probabilities. A naive way to find the top k probabilities is to compute all probabilities then select the top k. Instead, we approximate probabilities only to the degree needed to guarantee that (a) the top k answers are the correct ones, and (b) the ranking of these top k answers is correct. In our running example, we will run an

Rank	Director	p
1	Woody Allen	0.9998
2	Ralph Senensky	0.715017
3	Fred Olen Ray	0.701627
4	George Cukor	0.665626
5	Stewart Raffill	0.645483
	...	...

Figure 3.4: Top 5 query answers, out of 1415

approximation algorithm for many steps on the, say, top $k = 10$ answers, in order to identify them and rank them, but will run only a few steps on the remaining $1415 - 10 = 1405$ answers, and approximate their probabilities only as much as needed to guarantee that they are not in the top 10. This turns out to be orders of magnitude more efficient than the naive approach. The major challenge is that we do not know which tuples are in top 10 before we know their probabilities: the solution to this is the main technical contribution in this chapter.

The problem we address here is the following: We are given a SQL query and a number k, and we have to return to the user the k highest ranked answers sorted by their output probabilities. To compute the probabilities we use Luby and Karp's Monte Carlo simulation algorithm [101] (MC), which can compute an approximation to any desired precision. A naive application of MC would be to run it a sufficiently large number of steps on each query answer and compute its probability with high precision, then sort the answers and return the top k. In contrast, we describe here an algorithm called *multisimulation* (MS), which concentrates the simulation steps on the top k answers, and only simulates the others a sufficient number of steps to ensure that they are not in the top k. We prove that MS is theoretically optimal in a very strong sense: it is within a factor of two of a non-deterministic optimal algorithm, which magically "knows" how many steps to simulate each answer, and *no other* deterministic algorithm can be better. There are three variations of MS: computing the set of top k answers, computing and sorting the set of top k answers, and an *any time* algorithm, which outputs the answers in the order $1, 2, 3, \ldots, k, \ldots$ and can be stopped at any time. Our experiments show that MS exploits gracefully k (the running times are essentially linear in k) and that MS is dramatically more efficient than a naive application of MC.

3.1.1 Prior Art: Computing the probability of a DNF Formula

As we observed in the Preliminaries (Chapter 2), computing the probability of a tuple in the output is equivalent to computing the probability of a DNF formula. For an answer tuple t, this DNF formula is exactly $\lambda(t)$, which we can compute using the rules for lineage. From Proposition 2.3.5, we know that the resulting formula $\lambda(t)$ is a DNF, and so we now recall prior art that computes the probability that λ is satisfied using a Monte Carlo-style algorithm.

A Monte Carlo algorithm repeatedly chooses at random a possible world, and computes the truth value of the Boolean expression λ; the probability $p = \mu(\ \lambda(t)\ )$ is approximated by the frequency $\tilde{p}$ with which $\lambda(t)$ was true. Luby and Karp have described the variant shown in Algorithm 3.1.1.1, which has better guarantees than a naive MC. For our purposes the details of the Luby and Karp algorithm are not important: what is important is that, after running for N steps, the algorithm guarantees with high probability that p is in some interval $[a, b]$, whose width shrinks as N increases. Formally:

Theorem 3.1.1. *[101] Let $\delta > 0$ and define $\varepsilon = \sqrt{4m\log(2/\delta)/N}$, where m is the number of disjunctions in $\lambda(t)$ and N is the number of steps executed by the Luby and Karp algorithm. Let $a = \tilde{p} - \varepsilon$ and $b = \tilde{p} + \varepsilon$. Then the value p belongs to $[a, b]$ with probability $\geq 1 - \delta$, i.e.[1]:*

$$\mu(p \in [a, b]) \quad > \quad 1 - \delta \tag{3.1}$$

3.2 Top-k Query Evaluation using Multisimulation

We now describe our algorithm. We are given a query q and an instance J stored in a SQL database engine, and we have to compute the top k answers for q on J. Evaluation has two parts: (1) evaluating the SQL query in the engine and grouping the answer tuples to construct λ, (2) running a Monte Carlo simulation on each group in the middleware to compute the probabilities, then returning the top k probabilities. We describe here the basic algorithm, and discuss optimizations in the next section.

[1] In the original paper the bound is given as $|\ p - \tilde{p}\ | \leq \varepsilon p$, since $p \leq 1$ this implies our bounds. The bound in that paper is a stronger, relative bound. This is a technical point that we return to later in Chapter 4, but is unimportant for our discussion here.

Algorithm 3.1.1.1 Luby-Karp algorithm for computing the probability of a DNF formula $\lambda(t) = \bigvee_i t_i$ where t_i is a disjunct.

fix an order on the disjuncts: $t_1, t_2, \ldots, t_m$
$C := 0$
repeat
  Choose randomly one disjunct $t_i \in \lambda$
  Choose randomly a truth assignment $\boldsymbol{x}$ with probability conditioned on *"t_i is satisfied"*
  if forall $j < i$ $t_j(\boldsymbol{x})$ = `false` **then** $C := C + 1$
until N times
return $\tilde{p} = C/N$

3.2.1 Multisimulation (MS)

We model the problem as follows. We are given a set $\mathcal{G} = \{G_1, \ldots, G_n\}$ of n objects, with unknown probabilities $p_1, \ldots, p_n$, and a number $k \leq n$. Our goal is to find a set of k objects with the highest probabilities, denoted `TopK` $\subseteq \mathcal{G}$: we discuss below how to also sort this set. The way we observe the objects' probabilities is by means of a simulation algorithm that, after running N steps on an object G, returns an approximation interval $[a^N, b^N]$ for its probability p, with $a^N < b^N$ (we assume $a^N = b^N$ can never happen). We make the following four assumptions about the simulation algorithm and about the unknown probabilities:

Convergence : $\lim_{N\to\infty} a^N = \lim_{N\to\infty} b^N$.

Precision : $\forall N. p \in [a^N, b^N]$.

Progress : $\forall N. [a^{N+1}, b^{N+1}] \subseteq [a^N, b^N]$.

Separation : $\forall i \neq j,\ p_i \neq p_j$.

By the separation assumption `TopK` has a unique solution, i.e. there are no ties, and by the other three assumptions the solution can be found naively by a round robin algorithm.

In our setting each object G is a group of tuples representing a lineage formula λ, and its probability is $p = \mu(\lambda)$. The simulation algorithm is Luby-Karp. Only the first assumption holds strictly (convergence): we revisit below how to address the other three.

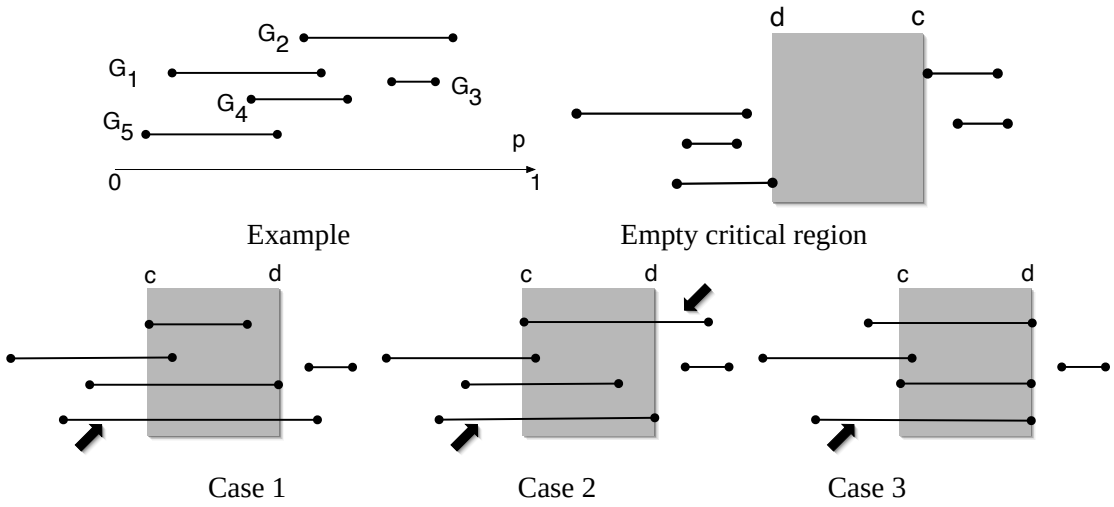


Figure 3.5: Illustration of MS; $k = 2$.

Intuition Any algorithm that computes TopK can only do this by running simulations on the objects. It initializes the intervals to $[a_1, b_1] = [a_2, b_2] = \ldots = [a_n, b_n] = [0, 1]$, then repeatedly chooses to simulate some G_i for one step. At each point in the execution, object G_i has been simulated N_i steps, and thus its interval is $[a_i^{N_i}, b_i^{N_i}] = [a_i, b_i]$ (we omit the superscript when it is clear). The total number of steps over all groups is $N = \sum_{i=1}^{n} N_i$. Consider the top left figure in Figure 3.5, where for $k = 2$. Here we have already simulated each of the five groups for a while: clearly G_3 is in the top 2 (it may be dominated only by G_2), although we don't know if it is 1st or 2nd. However, it is unclear who the other object in top 2 is: it might be G_1, G_2, or G_4. It is also certain that G_5 is not among the top 2 (it is below G_2, G_3).

Given two intervals $[a_i, b_i]$, $[a_j, b_j]$, if $b_i \leq a_j$ then we say that the first is *below*, and the second is *above*. We also say that the two intervals are *separated*: in this case we know $p_i < p_j$ (even if $b_i = a_j$, due to the "separation" assumption). We say that the set of n intervals is k-separated if there exists a set $T \subseteq \mathcal{G}$ of exactly k intervals such that any interval in T is above any interval not in T. Any algorithm searching for the TopK must simulate the intervals until it finds a k-separation (otherwise we can prove that TopK is not uniquely determined); in that case it outputs TopK $= T$. The cost of the algorithm is the number of steps N at termination.

Our golden standard will be the following nondeterministic algorithm, OPT, which is obviously optimal. OPT "knows" exactly how many steps to simulate G_i, namely N_i^{opt} steps, such that the following holds (a) the intervals $[a_1^{N_1^{opt}}, b_1^{N_1^{opt}}]$, ..., $[a_n^{N_n^{opt}}, b_n^{N_n^{opt}}]$ are k-separated, and (b) the sum $N^{opt} = \sum_i N_i^{opt}$ is minimal. When there are multiple optimal solutions, OPT chooses one arbitrarily. Clearly such an oracle algorithm cannot be implemented in practice. Our goal is to derive a *deterministic* algorithm that comes close to OPT.

Example 3.2.1 To see the difficulties, consider two objects G_1, G_2 and $k = 1$ with probabilities $p_1 < p_2$, and the current intervals (say, after simulating both G_1 and G_2 for one step) are $[a_1, b_1]$, $[a_2, b_2]$ such that $a_1 = p_1 < a_2 < b_1 < p_2 = b_2$. The correct top-1 answer is G_2, but we don't know this until we have separated them: all we know is $p_1 \in [a_1, b_1]$, $p_2 \in [a_2, b_2]$ and it is still possible that $p_2 < p_1$. Suppose we decide to simulate repeatedly only G_2. This clearly cannot be optimal. For example, G_2 may require a huge number of simulation steps before a_2 increases above b_1, while G_1 may take only one simulation step to decrease b_1 below a_2: thus, by betting only on G_2 we may perform arbitrarily worse than OPT, which would know to choose G_1 to simulate. Symmetrically, if we bet only on G_1, then there are cases when we perform much worse than OPT. Round robin seems a more reasonable strategy, i.e. we simulate alternatively G_1 and G_2. Here, the cost is twice that of OPT, in the following case: for N steps a_2 and b_1 move very little, such that their relative order remains unchanged, $a_1 < a_2 < b_1 < b_2$. Then, at the $N + 1$'th step, b_1 decreases dramatically, changing the order to $a_1 < b_1 < a_2 < b_2$. Round robin finishes in $2N + 1$ steps. The N steps used to simulate G_2 were wasted, since the changes in a_2 were tiny and made no difference. Here OPT chooses to simulate only G_1, and its cost is $N + 1$, which is almost half of round robin. In fact, no deterministic algorithm can be better than twice the cost of OPT. However, round robin is not always a good algorithm: sometime it can perform much worse than OPT. Consider n objects $G_1, \ldots, G_n$ and $k = 1$. Round robin may perform n times worse than OPT, since there are cases in which (as before) choosing the right object on which to bet exclusively is optimal, while round robin wastes simulation steps on all the n objects, hence its cost is $n \cdot N^{opt}$.

Notations and definitions Given n non-negative numbers $x_1, x_2, \ldots, x_n$, not necessarily distinct, let us define $top_k(x_1, \ldots, x_n)$ to be the k's largest value. Formally, given some permutation such that $x_{i_1} \geq x_{i_2} \geq \ldots \geq x_{i_n}$, top_k is defined to be x_{i_k}. We set $top_{n+1} = 0$.

Definition 3.2.2. *The* critical region, top objects, *and* bottom objects *are:*

$$\begin{aligned}(c,d) &= (top_k(a_1,\ldots,a_n), top_{k+1}(b_1,\ldots,b_n)) \\ T &= \{G_i \mid d \le a_i\} \\ B &= \{G_i \mid b_i \le c\}\end{aligned} \tag{3.2}$$

One can check that $B \cap \texttt{TopK} = \emptyset$ and $T \subseteq \texttt{TopK}$: e.g. $b_i \le c$ implies (by definition of c) that there are k intervals $[a_j, b_j]$ above $[a_i, b_i]$, which proves the first claim. Figure 3.5 illustrates four critical regions.

The important property of the critical region is that the intervals have a k-separation iff the critical region is empty, i.e., $c \ge d$, in which case we can return $\texttt{TopK} = T$. This is illustrated in the upper right of Figure 3.5, where the top 2 objects are clearly those to the right of the critical region. We therefore assume $c < d$ from now on. Call an object G_i a *crosser* if $[a_i, b_i]$ contains the critical region, i.e. $a_i \le c$, $d \le b_i$. There are always at least two crossers. Indeed, there are $k+1$ intervals $[a_i, b_i]$ such that $d \le b_i$. and at most $k-1$ of them may satisfy $c < a_i$; hence, the others (at least two) satisfy $a_i \le c$, and are crossers. Given a crosser $[a_i, b_i]$ we call it an *upper crosser* if $d < b_i$, a *lower crosser* if $a_i < c$, and a *double crosser* if it is both.

The Algorithm is shown in Algorithm 3.2.1.1. At each step it picks one or two intervals to simulate, according to three cases (see Fig 3.5). First, it tries a double crosser $[a_i, b_i]$; if there is none then it tries to find an upper crosser, lower crosser pair; if none exists then it means that either all crossers have the same left endpoint $a_i = c$ or all have the same right endpoint $d = b_i$. In either case there exists a maximal crosser, i.e., one that contains all other crossers: pick one and simulate it (there may be several, since intervals may be equal). After each iteration re-compute the critical region; when it becomes empty, stop and return the set T of intervals above the critical region. Based on our previous discussion the algorithm is clearly correct: i.e., it returns TopK when it terminates. From the **convergence** assumption it follows that the algorithm terminates.

Analysis We now prove that the algorithm is optimal within a factor of two of OPT, and, moreover, that no deterministic algorithm can be better.

At any point during the algorithm's execution we say that an interval $[a_i, b_i]$ *has slack* if $N_i < N_i^{opt}$. If it has slack then the algorithm can safely simulate it without doing worse than OPT. We

Algorithm 3.2.1.1 The Mulisimulation Algorithm

MS_TopK$(\mathcal{G}, k)$: /* $\mathcal{G} = \{G_1, \ldots, G_n\}$ */
Let $[a_1, b_1] = \ldots = [a_n, b_n] = [0, 1]$, $(c, d) = (0, 1)$
while $c \leq d$ **do**
  Case 1: exists a double crosser: simulate it one step
  Case 2: exists an upper crosser and a lower crosser: simulate both one step
  Case 3: otherwise: pick a maximal crosser, simulate it one step
  Update (c, d) using Eq.(3.2)
return TopK $= T = \{G_i \mid d \leq a_i\}$

have:

Lemma 3.2.3. *Let $[a_i, b_i]$ be a crosser. Then, in all cases below, $[a_i, b_i]$ has slack: (1) If it is an upper crosser and is not in the top k. (2) If it is a lower crosser and is in the top k. (3) If it is a double crosser. (4) If it contains all crossers (i.e. it is a maximal crosser).*

Proof. To see (1), note that OPT must find k intervals above i; but since $[a_i^{N_i}, b_i^{N_i}]$ is an upper crosser, there are at most $k-1$ $b_j^{N_j}$'s such that $b_j^{N_j} > b_i^{N_j}$; hence, OPT can find at most $k-1$ intervals (namely the same ones, at most) that are above $b_i^{N_i}$, i.e. $a_j^{N_j^{opt}} > b_i^{N_i}$, because $a_j^{N_j^{opt}} < b_j^{N_j}$ (due to the progress assumption). It follows that OPT must simulate i at least one more step than N_i to bring $b_i^{N_i^{opt}}$ below $b_i^{N_i}$ in order to separate it from the top k. (2) and (3) are similar. To prove (4), we assume that the interval i is in TopK: the other case is symmetric. Consider the $k+1$ intervals that have $b_j \geq d$: at least one, say $[a_j, b_j]$, must be not in TopK, and OPT must separate them by proving that $[a_j, b_j]$ is below $[a_i, b_i]$. But $a_i \leq a_j$ because either $[a_j, b_j]$ is included in $[a_i, b_i]$, or $[a_j, b_j]$ is not a crosser (hence $a_i \leq c \leq a_j$). Hence, to separate them, OPT must either reduce $[a_j, b_j]$ to a point or further simulate $[a_i, b_i]$. But since we assume that an MC algorithm cannot return a point interval (i.e. $a^N < b^N$ forall N), OPT must simulate $[a_i, b_i]$. □

Theorem 3.2.4. *(1) The cost of algorithm* **MS_TopK** *is $< 2N^{opt}$. (2) For any deterministic algorithm computing the top k and for any $c < 2$ there exists an instance on which its cost is $\geq cN^{opt}$.*

Proof. To prove (1), notice that at each step the algorithm simulates one or two intervals: it suffices to prove that at least one of them has slack[2]. There are three cases. First, a double crosser is

[2]This shows that the cost is $\leq 2N^{opt}$; to prove $< 2N^{opt}$ one notices that at least one iteration simulates a single interval, with slack.

simulated: clearly it has slack. Second, an upper and a lower crosser are simulated: in order for both not to have slack we must have one is in the top k and the other is not in the top k; but in that case OPT must simulate at least one of them, since they are not separated yet, hence one of them does have slack after all. Third, there are only upper or only lower crossers and we simulate the largest one: we have seen that this also has slack.

The main idea for (2) is in Example 3.2.1. Given a deterministic algorithm A, we construct a family of instances indexed by $m > 1$ (and that depend on A) such that the optimal has cost N_m^{opt} and A can do better than $2N_m^{opt}$. The instance contains two intervals I and J. The intuition is that one of these two intervals will separate at "time" $2m$, but not before. Initially, $I = \left[0, \frac{2}{3}\right]$ and $J = \left[\frac{1}{3}, 1\right]$. Suppose that the algorithm spends s_I on simulating interval I and s_J simulating interval J where $s_I + s_J < 2m$. Then, the intervals will shrink to $I = \left[\frac{s_I}{6m}, \frac{2}{3}\right]$, similarly $J = \left[\frac{1}{3}, 1 - \frac{s_J}{6m}\right]$. Since $s_I + s_J \leq 2m$, this means that the intervals will overlap for the entire time. To construct the m-th instance, let the algorithm run for m steps and shrink the intervals as above. Then, choose which ever interval the algorithm has simulated least and make it collapse. More precisely, assume that interval is I: if I was simulated least, then on the next iteration of I we will set $I = \left[\frac{s_I}{6m}, \frac{s_i+1}{6m}\right]$. Notice that $s_I < m$. OPT only simulates I which has cost $N_m^{opt} = s_I < m$, while A pays cost at least $2m$ on this instance. A symmetric argument holds if the algorithm chooses J less frequently. Notice that since the algorithm is deterministic, it is only a function of the intervals. □

Corollary 3.2.5. *Let* **A** *be any deterministic algorithm for finding* TopK*. Then (a) on any instance the cost of* **MS_TopK** *is at most twice the cost of* **A**, *and (b) for any $c < 1$ there exists an instance where the cost of* **A** *is greater than c times the cost of* **MS_TopK**.

3.2.2 Discussion

Variations and extensions In query answering we need to compute the top k answers *and* to sort them. The following variation of MS, which we call **MS_RankK**, does this. First, compute the top k, $T_k =$ **MS_TopK**$(\mathcal{G}, k)$. Next, compute the following sets, in this sequence:

$$
\begin{aligned}
T_{k-1} &= \textbf{MS\_TopK}_{ni}(T_k, k-1) \\
T_{k-2} &= \textbf{MS\_TopK}_{ni}(T_{k-1}, k-2) \\
&\dots \\
T_1 &= \textbf{MS\_TopK}_{ni}(T_2, 1)
\end{aligned}
$$

At each step we have a set T_j of the top j answers, and we compute the top $j-1$: this also identifies the j'th ranked object. Thus, all top k objects are identified, in reverse order. Here $\textbf{MS\_TopK}_{ni}$ denotes the algorithm **MS_TopK** without the first line: that is, it does not initialize the intervals $[a_i, b_i]$ but continues from where the previous multisimulation left off. This algorithm is also optimal: It is straightforward to prove a theorem similar to 3.2.4.

The second variation is an *any-time* algorithm, which computes and returns the top answers in order, without knowing k. The user can stop any time. The algorithm starts by identifying the top element $T_1 = \textbf{MS\_TopK}(\mathcal{G}, 1)$, then it finds the remaining groups in decreasing order: $T_{j+1} = \textbf{MS\_TopK}(B_j, 1)$, where $B_j = \mathcal{G} - (T_1 \cup \ldots \cup T_j)$. Note that for $k > 1$ this algorithm is *not* optimal in finding the top k elements; its advantage is in its any-time nature. Also, it prevents the semi-join optimization discussed below, which requires knowledge of k.

Revisiting the assumptions Precision holds for any MC algorithm, but only in a probabilistic sense. For example after running Luby-Karp's algorithm for N steps, $\mu(p \in [a^N, b^N]) > 1 - \delta_1$. The choice of the *confidence* δ_1 affects the convergence rate: $b^N - a^N = 2\sqrt{4m \log(2/\delta_1)/N}$, where m is the size of the group. In our context the user chooses a global parameter δ and requires that *all* n groups be precise with confidence δ. Assuming equal confidences, the system sets δ_1 for each group to δ/n, since it implies $(1-\delta_1)^n \geq 1 - \delta$. Still, since it appears under log, we can choose *very small* values for δ without affecting significantly the running time (N), hence precision holds for all practical purposes. The separation assumption is more problematic, since in practice probabilities are often equal or very close to each other. Here we simply rely on a second parameter $\varepsilon > 0$: when the critical region becomes less than ε, we stop and rank the uncertain groups based

on the midpoints of their intervals. Progress as stated, does not hold for our Monte Carlo simulation technique: After a large number of steps, say n, our estimate may have jumped by a factor of n^{-1}; this means that its "confidence interval" is like $n^{-1} + n^{-1/2}$ – which is outside the bounding interval. Instead, what does hold is the weaker statement that after a negligible number of steps with respect to OPT, the sequence of intervals are again nested (with high probability). This allows us to show that MS takes $2OPT + o(OPT)$ steps (see Appendix A.1). The choice of ε also affects running time and precision/recall. We discuss the system's sensitivity on δ and ε in Section 3.4.

Finally, note that our restriction that the intervals never collapse (i.e., $a^N < b^N$ forall N) is important. This is always true in practice (for any MC algorithm). As a pure theoretical observation we note here that without this assumption the proof of Lemma 3.2.3 (4) fails and, in fact, no deterministic algorithm can be within a constant factor of OPT. Consider searching for the top $k = 1$ of n objects; all n intervals start from the initial configuration $[0, 1]$. OPT picks the winner object, whose interval, after one simulation step, collapses to [1,1]: OPT finishes in 1 step, while any deterministic algorithm must touch all n intervals at least one.

Further Improvements One may wonder if our adversarial model in which intervals may shrink at arbitrary, unpredictable rates is too strong. In theory it may be possible to design an algorithm that finds `TopK` by exploiting the specific rates at which the intervals shrink (see the bounds in Th. 3.1.1). However, note that this will result in at most a factor of 2 improvement over the MS algorithm, due to Corollary 3.2.5.

3.3 Optimizations

We present two optimizations: the first reduces the number of groups to be simulated using a simple pruning technique, the second reduces the sizes of the groups by pushing more of the processing from the middleware to the engine. Both techniques are provably correct in that they are guaranteed to preserve the query's semantics.

Pruning The following are two simple upper and lower bounds for the probability of a group $\lambda(t) = \phi_1 \vee \cdots \vee \phi_n$ for any tuple t

$$\max_{i=1}^{m} \mu(\phi_i) \leq \quad \mu(\lambda(t)) = \mu(\bigvee_{i=1,\ldots,n} \phi_i) \quad \leq \sum_{i=1}^{m} \mu(\phi_i)$$

They can be computed easily and allow us to initialize the *critical region* using Eq. 3.2 and to prune some groups before even starting MS. As an improvements, when there are no pairs of disjoint tuples in the group (which is a condition that can be checked statically) then the upper bound can be tightened to $1 - \prod_i (1 - \mu(\phi_i))$.

Safe Subqueries Sometimes the probabilities can be pushed to the engine, by multiplying probabilities (when the tuples are independent) or by adding them (when the tuples are disjoint). This can be achieved by running a SQL query, over a subset $\overline{\mathtt{R}}' \subseteq \overline{\mathtt{R}}$ of the tables in the original query `q`, like the following (here $\overline{\mathtt{R}}' =$ `R1, R2, R2`):

```
sq = SELECT B̄′, AGG(R1^p.p*R2^p.p*R3^p.p) as p
     FROM R1^p, R2^p, R3^p WHERE C GROUP-BY B̄′
```

where `AGG` is either `sum` or `prod_1_1`:

$$\begin{aligned} \mathtt{sum}(p_1, \ldots, p_m) &= \sum_i p_i \\ \mathtt{prod\_1\_1}(p_1, \ldots, p_m) &= 1 - \prod_i (1 - p_i) \end{aligned}$$

The optimization works like this. Given the query `q` choose a subset of its tables $\overline{\mathtt{R}}' \subseteq \overline{\mathtt{R}}$, and some set of attributes $\overline{\mathtt{B}}'$ (which must include all attributes on which the relations $\overline{\mathtt{R}}'$ join with the other relations). Then construct a subquery like `sq` above, and use it as a subexpression in `q` as it were a normal BID table[3], with probability given by `p`, and its possible-worlds key given by a certain subset $\overline{\mathtt{S}}$ of $\overline{\mathtt{B}}'$.

[3]We will later in Chapter 5 see a more general condition called *representability* that establishes the exact conditions for a table to be a BID table, which are more general than the sufficient conditions that we discuss here.

Three conditions must be met for this rewriting to be correct. (1) The tuple probability p computed by AGG must be correct, (2) in the output, tuples having the same value of $\overline{S}$ must be disjoint tuples and tuples having different values of $\overline{S}$ must be independent tuples, and (3) each such probability must be independent of all the other tuple in the original query that it joins with. Recall that Key(R) denotes the set of key attributes for the possible worlds for R.

To check (1) we have the following:

Proposition 3.3.1. *Consider the query sq above. Let Attr(R) denote the attributes of relation R (does not include the p attribute, which technically belongs only to R^p) and Attr(sq) denote the union of Attr(R) for all relations R in sq.*

1. *If AGG is sum then p is computed correctly iff $\exists R \in \overline{R}'$ s.t. Key(R) $\subseteq \overline{B}'$ and Attr(sq) $- \overline{B}' \subseteq$ Attr(R).*

2. *If AGG is prod_1_1 then p is computed correctly iff $\forall R \in \overline{R}'$, Attr(sq) $- \overline{B}' \subseteq$ Key(R).*

To check (2) we have the following:

Proposition 3.3.2. *Consider the query sq above.*

1. *Two output tuples having the same values of $\overline{S}$ are disjoint events iff $\exists R \in \overline{R}'$ s.t. Key(R) $\subseteq \overline{S}$ and $\overline{B}' - \overline{S} \subseteq$ Attr(R).*

2. *Two output tuples having different values of $\overline{S}$ are independent events iff $\forall R \in \overline{R}'$, $\overline{B}' - \overline{S} \subseteq$ Key(R).*

Finally, to check (3) we need to check that the relations used by sq do not occur again the rest of the query q.

Example 3.3.3 Consider three probabilistic tables:

AmazonHighReviews^p(asin, reviewer, p)

TitleMatch^p(asin, imdbid, p)

IMDBHighRatedFilms^p(imdbid, p)

with possible worlds keys

Key(AmazonHighReviews) = {asin, reviewer}

Key(TitleMatch) = {asin}

Key(IMDBHighRatedFilms) = {imdbid}

Note that AmazonHighReviews and IMDBHighRatedFilms contain only independent tuples. Consider the query q:

```
q = TOP 5 SELECT DISTINCT A.reviewer
    FROM AmazonHighReviews A,
         TitleMatch T, IMDBHighRatedFilms I
    WHERE A.asin = T.asin and T.imdbid = I.imdbid
```

The query can be optimized by observing that the following subquery is a safe subquery:

```
sq = SELECT T.asin, sum(T.p * I.p) as p
     FROM TitleMatch^p T, IMDBHighRatedFilms^p I
     WHERE T.imdbid = I.imdbid
     GROUP BY T.asin
```

The output of this subquery is a table $\texttt{Tmp}^p\texttt{(asin, p)}$ that can be treated as a base probabilistic table with possible world key asin and probability attribute p. To see why, let us verify that this subquery satisfies the three conditions for safe subquery:

- For condition (1), we use Prop. 3.3.1(1). Here $\overline{\texttt{B}}' = \{\texttt{asin}\}$ and Attr(sq) = {asin, imdbid}. We see that $\texttt{Key(TitleMatch)} \subseteq \overline{\texttt{B}}'$ and $\texttt{Attr(sq)} - \overline{\texttt{B}}' \subseteq$ Attr(TitleMatch), so the condition is met.

- For condition (2), we use Prop. 3.3.2. Here, $\overline{\texttt{S}} = \{\texttt{asin}\}$ since we are claiming that asin is the key for Tmp. Prop. 3.3.2(2) holds trivially because $\overline{\texttt{B}}' - \overline{\texttt{S}} = \emptyset$. Prop. 3.3.2(1) holds because $\texttt{Key(TitleMatch)} \subseteq \overline{\texttt{S}}$.

- Condition (3) holds because all event tables outside Tmp are distinct from those inside.

Probabilistic Table Name	#Tuples	#exclusive tuples	
		Avg.	Max
MovieToAsin	339095	4	13
AmazonReviews	292680	1	1
ActorMatch	6758782	21	2541
DirectorMatch	18832	2	36
UsenetMatch	134803	5	203
UsenetReview	3159	1	3159
ActivityData	2614480	3	10
HMM	100	10	10

Figure 3.6: Three Case Studies of Imprecisions

Query name	# of groups (n)	Avg group size m		Max group size		# of prob. tables (m)
		no SP	SP	no SP	SP	
SS	33	20.4	8.4	63	26	2
SL	16	117.7	77.5	685	377	4
LS	3259	3.03	2.2	30	8	2
LL	1415	234.8	71.0	9088	226	4

Figure 3.7: Query Stats w/o and w/ S(afe) P(lan)

Having verified that the subquery is indeed safe, we rewrite the query `q` by making `sq` a subquery:

```
q_safe-plan = TOP 5 SELECT DISTINCT A.reviewer
              FROM AmazonHighReviews A, sq Tmp
              WHERE A.asin = Tmp.asin
```

Thus, the table `Tmp(asin,p)` is computed inside the engine, and treated like a base query by MS. The rest of MS remains unchanged. The new query has the same number of groups as the original query, but each group is much smaller since some of the probabilistic computation has been pushed in the engine.

3.4 Experiments

In this section we evaluate our approach experimentally. We address five questions: (1) what is the scale of probabilistic databases when modeling imprecisions; (2) how does our new query evaluation method compare to the current state of the art; (3) how effective is the multisimulation (MS) over a naive approach (4) how effective are the optimizations; and (5) how sensitive is the system's performance on the choice of δ and ε.

Setup Our experiments were run on a dual processor Intel Xenon 3GHz Machine with 8G RAM and 2 400GB disks. The operating system used was Linux with kernel version 2.6.12 high-mem build. The database was DB2 UDB Trial Edition, v. 8.2. Due to licensing restrictions DB2 was only one able to use one of the cores. Indexes and configuration parameters such as buffer pools were tuned by hand.

Methodology For each running time we perform the experiment 5 times, dropping the highest and the lowest and average the remaining three runs. The naive simulation method was capped at 20 minutes. In between each experiment, we force the database to terminate all connections. The same experiments was not run repeatedly to minimize caching effects but the cache was allowed to be warm. In the precision/recall experiments, the precision and recall are defined as the fraction of the top k answers returned by method being evaluated that overlap with the "correct" set of top k answers. In order to compute the latter we had to compute the exact tuple probabilities, which is intractable. For that we used the approximate values returned by the simulation algorithm with very low settings for ε and δ: $\varepsilon = 0.001$ and $\delta = 0.01$.

3.4.1 Case Studies

In an empirical study we modeled imprecisions in three application domains. The first integrates the IMDB movie database with reviews from Amazon, as described in a simplified form in Sec. 3.1, and the sources of imprecisions are fuzzy object matches (for titles, actors, and directors), and the confidence in the amazon reviews ("how many people found this review useful"). The second application integrates IMDB with reviews collected from a USENET site[4]. These reviews were in free text and we had to use information extraction techniques to retrieve for each review (a) the movie and (b) the

[4] `ftp://ftp.uu.net/usenet/rec.arts.movies.reviews/`

rating. The imprecisions here were generated by information extraction tools. In the third application we used human activity recognition data obtained from body-worn sensors [110]. The data was first collected from eight different sensors (accelerometer, audio, IR/visible light, high-frequency light, barometric pressure, humidity, temperature, and compass) in a shoulder mounted multi-sensor board, collected at a rate of 4 per second, then classified into into $N = 10$ classes of human activity $A^1, A^2, \ldots, A^N$, one for each subject and each time unit. The classes were: riding elevator up or down, driving car, riding bicycle, walking stairs up or down, jogging, walking, standing, sitting. The imprecisions here come from the classification procedure, which results in probability distribution on the N activities.

Figure 3.6 shows brief summaries of the probabilistic data in each of these applications. Each required between two and four base probabilistic tables, and between one to three SQL views for complex probabilistic correlations. In addition to the probabilistic data IMDB had some large deterministic tables (over 400k movies, 850k actors, and 3M casts, not shown in the figure), which are part of the query processor's input in the experiments below, hence they are important for our evaluation.

3.4.2 Query Performance

We report below measurements only from the first data set (IMDB-Amazon integration), which was the largest and richest. The processor's performance was mostly affected by two query parameters: the number of groups (denoted *n* in Sec. 3.2) and the average size of each group. In additional experiments (not shown) we noticed that the performance was less affected by the number of probabilistic tables in the query (*m* in Sec. 3.2), which roughly corresponds to the number of sources of evidence in the imprecise data.

By choosing each parameter to be small (S) or large (L) we obtained four classes of queries denote SS, SL, LS, and LL respectively; we chose one query form each class, and show it in Figure 3.7. The queries are:

SS In which years did *Anthony Hopkins* appear in a highly rated movie? (Our system returns the top answer 2001, the year he was in Hannibal)

SL Find all actors who were in Pulp Fiction who were in two very bad movies in the five years before Pulp Fiction. (Top 2 Answers: Samuel L Jackson and Christopher Walken)

LS Find all directors who had a low rated movie between 1980 and 1989. (Top 2 Answers: Richard C. Sarafian for Gangster Wars and Tim King for Final Run)

LL Find all directors who had a low rated drama and a high rated comedy less than five years apart. (Top Answer: Woody Allen)

Unless otherwise stated, the confidence and precision parameters were $\varepsilon = .01$, $\delta = .01$, and the multisimulation algorithm run was **MS_RankK** (Sec. 3.2.2), which finds the top k and sorts them.

Comparison with Other Methods The state of the art in query evaluation on probabilistic databases is to either compute each query answer exactly, using a complete Monte Carlo simulation (we call this method naive (N)), or to approximate the probabilities using some strategies [108] by ignoring their correlations. The first results in much larger running times than multisimulation (MS): see Figure 3.8 (a) (note the logarithmic scale): the naive method timed out for the LS and LL queries. The approximation method is much faster than MS, but results in lower precision/recall, due to the fact that it ignores correlations between imprecisions: this is shown in Figure 3.8 (b). Note that, unlike a Monte Carlo simulation, where precision and recall can be improved by running longer, there is no room for further improvement in the approximate method. Note that one of the queries (LS) flattened at around 60% precision/recall. The queries that reached 100% did so only when k reached the total number of groups: even then, the answers are much worse then it looks since their order is mostly wrong. This clearly shows that one cannot ignore correlations when modeling imprecisions in data.

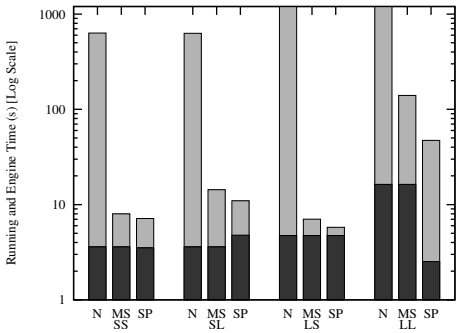
Analysis of Multisimulation The main idea of the multisimulation algorithm is that it tries to spend simulation steps on only the top k buckets. We tested experimentally how the total number of simulation steps varies with k, and in which buckets the simulation steps are spent. We show here the results for SS. Figure 3.8 (c) shows the total number of simulation steps as a function of k, both for the `TopK` algorithm (which only finds the top k set without sorting it) and for the `RankK` algorithm (which finds *and* sorts the top k set). First, the graph clearly shows that `RankK` benefits from low values of k: the number increases linearly with k. Second, it shows that, for

`TopK`, the number of steps is essentially independent on k. This is because most simulation steps are spent at the separation line between the top k and the rest. A deeper views is given by the graph in Figure 3.8 (d), which shows for each group (bucket) how many simulation steps were spent, for $k = 1, 5, 10, 25$, and 50. For example, when $k = 1$ most simulation steps are spent in buckets 1 to 5 (the highest in the order of the probability). The graph illustrates two interesting things: that `RankK` correctly concentrates most simulation steps on the top k buckets, and that, once k increases beyond a given bucket's number, the number of simulation steps for that bucket does not further increase. The spikes in both graphs correspond to clusters of probabilities, where MS had to spend more simulation steps to separate them. Figure 3.8 (e) shows the effect of k on the measured running time of each query. As expected, the running time scales almost linearly in k. That is, the fewer answer the user requests, the faster they can be retrieved.

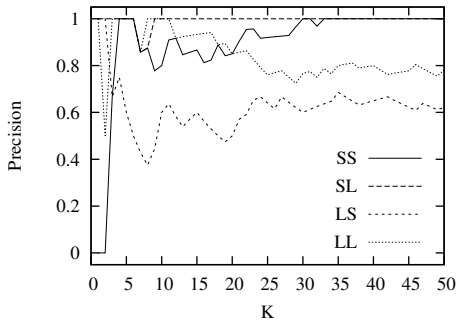
Effectiveness of the Optimizations We tested both optimizations: the semijoin pruning and safe query rewriting. The semijoin pruning was always effective for the queries with a large number of buckets (LS, LL), and harmless for the other two. We performed the pruning in the middleware, and the additional cost to the total running time was negligible. The safe-plan rewriting (SP) is more interesting to study, since it is highly non-trivial. Figure 3.8 (a) shows significant improvements (factors of 3 to 4) in the running times when the buckets are large (SL, LL), and modest improvements in the other cases. The query time in the engine differed, since now the queries issued are different: in one case (SL) the engine time was larger. Figure 3.7 shows how the SP optimization affects the average group size: this explains the better running times.

Sensitivity to Parameters Finally, we tested the system's sensitivity to the parameters δ and ε (see Sec. 3.2.2). Recall that the theoretical running time is $O(1/\varepsilon^2)$ and $O(\log(1/(n\delta))$. Figure 3.8 (f) shows both the precision/recall and the total running time as a function of $1 - \varepsilon$, for two queries: LL and LS; $k = 20$, $\delta = 0.01$, and SP is turned off. The running time are normalized to that of our golden standard, $1 - \varepsilon = 0.99$. As $1 - \varepsilon$ increases, the precision/recall quickly approaches the upper values, while the running time increases too, first slowly, then dramatically. There is a price to pay for very high precision/recall (which is what we did in all the other experiments). However, there is some room to tune $1 - \varepsilon$: around 0.9 both queries have a precision/recall of 90%-100% while the running time is significantly less than the golden standard. The similar graphs for δ differ, and is much more boring: the precisions/recall reaches 1 very fast, while the running time is almost

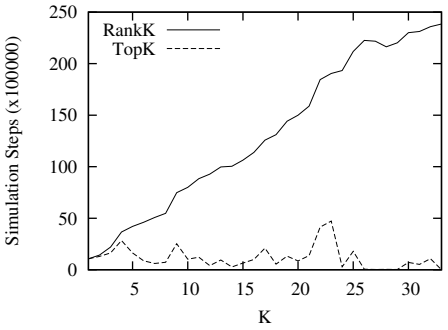
independent on δ. (The graphs look almost like two horizontal lines.) We can choose δ in a wide range without degrading either precision/recall or performance.



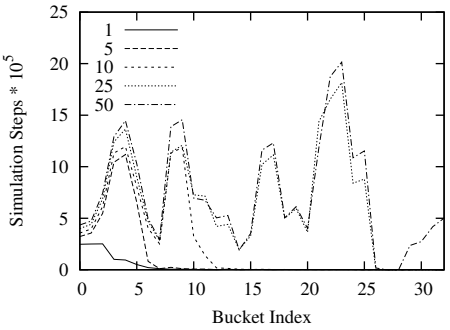
(a) Running times N(aive), MS, and S(afe) P(lan) $[k = 10, \varepsilon = .01, \delta = .01]$



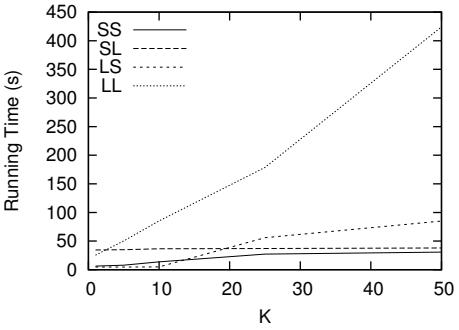
(b) Precision/Recall for naive strategies



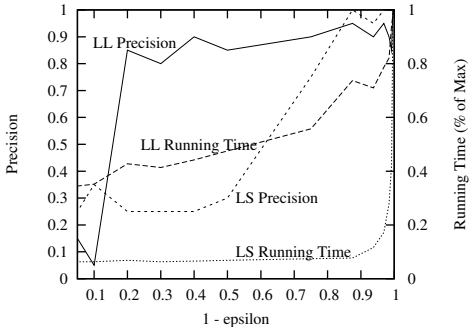
(c) Total number of simulation steps for query SS



(d) Number of simulation steps per bucket for query SS



(e) Effect of K on Running Time



(f) Effect of ϵ on Precision and Running Time

Figure 3.8: Experimental Evaluation

Chapter 4

QUERY-TIME TECHNIQUE II: EXTENSIONAL EVALUATION FOR AGGREGATES

In traditional databases, aggregation is a key technique to summarize a single large database instance. In probabilistic databases, we need to summarize *many* large instances (possible worlds). Intuitively, this suggests that aggregation may actually be a more fundamental operation on probabilistic data than even over standard, deterministic relational data. In this section, we both tackle the algorithmic challenge of evaluating (and approximating) aggregate queries on a probabilistic databases, and also, discuss the limits of any approach.

In this chapter, we study *HAVING queries* which are inspired by the HAVINGclause in SQL; more precisely, a HAVING is a conjunctive Boolean queries with an aggregation predicate, e.g., is the MAX greater than 10?. Informally, the technical highlight of this chapter is a trichotomy result: we show that for each HAVING query Q the complexity of evaluating Q falls into one of three categories: (1) The exact evaluation problem has P-time data complexity. In this case, we call the query *safe*. (2) The exact evaluation problem is ♯P-hard, but the approximate evaluation problem has (randomized) P-time data complexity. More precisely, there exists an FPTRAS for the query. In this case, we call the query *apx-safe*. (3) The exact evaluation problem is ♯P-hard, and the approximate evaluation problem is also hard (no FPTRAS likely exists). We call these queries *hazardous*.

4.1 Motivating Scenario

In SQL, aggregates come in two forms: *value aggregates* that are returned to the user in the SELECT clause (e.g., the MAX price) and *predicate aggregates* that appear in the HAVING clause (e.g., is the MAX price greater than $10.00?). In this section, we focus on positive conjunctive queries with a single predicate aggregate that we call HAVING queries. Prior art [26, 97] has defined a semantic for *value aggregation* that returns the expected value of an aggregate query (e.g., the expected MAX price) and has demonstrated its utility for OLAP-style applications. In this section, we propose a

Item	Forecaster	Profit	P
Widget	Alice	$−99K	0.99
	Bob	$100M	0.01
Whatsit	Alice	$1M	1

`Profit`(Item;Forecaster,Profit;P)

```
SELECT SUM(PROFIT)
FROM PROFIT
WHERE ITEM='Widget'
```

(a) Expectation Style

```
SELECT ITEM
FROM PROFIT
WHERE ITEM='Widget'
HAVING SUM(PROFIT) > 0.0
```

(b) `HAVING` Style

Figure 4.1: A probabilistic database with a `Profit` relation that contains the profit an analyst forecasts for each item sold. Prior Art [97] has considered a semantic similar to the query in (a), which returns the expected value of an aggregate. In contrast, we study queries similar to (b) which computes the probability of a `HAVING` style predicate, e.g., that the `SUM` of profits exceeds a value (here, 0.0).

complementary semantic for predicate aggregates inspired by `HAVING` (e.g., what is the *probability* that the `MAX` price is bigger than $10.00?). We illustrate the difference between the approaches with a simple example:

Example 4.1.1 Figure 4.1 illustrates a probabilistic database that contains a single relation, `Profit`. Intuitively, a tuple in `Profit` records the profit that one of our analysts forecasts if we continue to sell that item. We are not certain in our prediction, and so `Profit` records a confidence with each prediction. For example, Alice is quite sure that we will lose money if we continue selling widgets; this is captured by the tuple (*Widget*, *Alice*, $ − 99K, 0.99) in `Profit`. Intuitively, 0.99 is the marginal probability of the fact (*Widget*, *Alice*, $ − 99k).

An example of a value aggregate is shown in Figure 4.1(a). In this approach, the answer to an aggregation query is the *expected value of the aggregate function*. Using linearity of expectation, the value of the query in Figure 4.1(a) is 100M * 0.01 + −99K * 0.99 ≈ 900K. Intuitively, this large value suggests that we should continue selling widgets because we expect to make money. A second approach (that we propose and study in this chapter), is akin to `HAVING` style aggregation in standard SQL. An example is the query in Figure 4.1(b) that intuitively says: *"What is the probability that we will make a profit?"*. The answer to this query is the probability that the value of the `SUM` is greater than 0. Here, the answer is only 0.01: this small probability tells us that we should stop selling widgets or risk going out of business.

Our technical starting point is the observation that we can evaluate a query q with an aggregate α on a deterministic database using a two step process: (1) annotate the database with values from some semiring, S_α, e.g., if α = `COUNT`, then we can take S_α to be the natural numbers, and (2) propagate these annotations during query processing (using the rules in Green *et al.* [81]). In this scheme, each tuple output by the query is annotated with a value in the semiring S_α that is exactly the *value* of the aggregate, e.g., the `COUNT` of the tuples returned by q. Thus, it is easy to check if the `HAVING` query is true: simply test the predicate aggregate on the value returned by the query, e.g., is the `SUM` returned by the query greater than 0? If the answer is yes, return true.

To evaluate aggregate queries on probabilistic databases, we generalize this approach. On a probabilistic database, the output of an aggregate query Q is described by a *random variable*, denoted s_Q, that takes values in S_α. A `HAVING` query Q whose predicate is, say, `COUNT`$(*) < k$, can be computed over a probabilistic database in two stages: (1) compute the distribution of the random variable, s_Q; and (2) apply a *recovery function* that computes the probability that $s_Q < k$, i.e., sum over all satisfying values of s_Q. The cost of this algorithm depends on the space required to represent the random variable s_Q, which may be exponential in the size of the database. This cost is prohibitively high for many applications[1]. In general, this cost is unavoidable, as prior art has shown that for `SELECT-PROJECT-JOIN` (SPJ) queries (without `HAVING`), computing a query's probability is ♯P-Complete[2] [46, 78].

Although evaluating general SPJ queries on a probabilistic database is hard, there is a class of SPJ queries (called *safe queries*) that can be computed efficiently and exactly [46, 135]. A safe query has a relational plan P, called a *safe plan*, that is augmented to compute the output probability of a query by manipulating the marginal probabilities associated with tuples. The manipulations performed by the safe plan are standard multiplications and additions. These manipulations are correct because the safe plan "knows" the correlations of the tuples that the probabilities represent, e.g., the plan only multiplies probabilities when the events are independent. To generalize safe plans

[1] A probabilistic database represents a distribution over standard, deterministic instances, called *possible worlds* [63]. A probabilistic database with n tuples can encode 2^n possible worlds, i.e., one for each subset of tuples. We defer to Section 4.2.1 for more details.

[2] ♯P defined by Valiant [162] is the class of functions that contains the problem of counting the number of solutions to NP-Hard problems (e.g., ♯3-SAT). Formally, we mean here that there is a polynomial reduction from a ♯P-Hard problem, and to any problem in ♯P. Since technically, the query evaluation problem itself is not in ♯P.

to compute HAVING queries, we provide analogous operations for semiring random variables. First, we describe *marginal vectors* that are analogous to marginal probabilities: a marginal vector is a succinct, but lossy, representation of a random variable. We then show that the operation analogous to multiplying marginal probabilities is a kind of *semiring convolution*. Informally, we show that substituting multiplications with convolutions is correct precisely when the plan is safe.

As we show, the running time of safe plans with convolutions is proportional to the number of elements in the semiring, S_α. Thus, to compute HAVING queries with an aggregate α efficiently, we need S_α to be small, i.e., S_α should contain at most polynomially many elements in the size of the instance. This condition is met when the aggregate α is one of {EXISTS, MIN, MAX, COUNT}. For $\alpha \in$ {SUM,AVG, COUNT(DISTINCT)}, the condition is not met. In these cases, our algorithm is efficient only for a restricted type of safe plans that we call α-*safe*. For α-safe plans, a HAVING query with α can be computed efficiently and exactly. Further, we show that α-safe plans capture tractable exact evaluation for queries without self joins[3]. More precisely, for each aggregate $\alpha \in$ {EXISTS, MIN, MAX, COUNT, SUM, AVG, COUNT(DISTINCT) }, there is a dichotomy for queries without self joins: Either (1) Q is α-safe, and so has a P-time algorithm, or (2) Q is not α-safe and evaluating Q exactly is ♯P-Hard. Further, we can decide whether a query is α-safe in P-time.

Exact evaluation is the gold standard, but in many applications, *approximately* computing probabilities suffices. For example, if the input probabilities are obtained heuristically, then computing the precise value of the output probability may be overkill. Alternatively, even if the probabilities are obtained precisely, a user may not care about the difference between a query that returns a probability score of .9 versus .90001; moreover, as we argued in Chapter 3 such precision may be sufficient to rank the answers. Leveraging these observations, we show that there are some queries that can be efficiently *approximated*, even though they are not α-safe (and so cannot be computed exactly). More precisely, we study when there exists a *Fully Polynomial Time Randomized Approximation Scheme* (FPTRAS) for approximating the value of a HAVING query[4]. Our key result is that there is a second dichotomy for approximate evaluation for queries without self joins: Either (1) an

[3] A self join is a join between a relation and itself. The query $R(x,y), S(y)$ does not have a self join, but $R(x,y), R(y,z)$ does.

[4] An FPTRAS can be thought of as a form of sampling that is guaranteed to rapidly converge and so is efficient. We defer to Definition 4.6.1 for formal details.

approximation scheme in this chapter can approximate a HAVING query efficiently, or (2) there is no such efficient approximation scheme. Interestingly, we show that the introduction of self joins *raises* the complexity of approximation: we show a stronger inapproximability result for queries involving self joins.

In general, the complexity of evaluating a HAVING query Q depends on the predicate that Q uses. More precisely, the hardness depends on both the aggregate function, α, and the comparison function, θ, which together are called an *aggregate-test pair*, e.g., in Figure 4.1(b) the aggregate-test pair is $(\texttt{COUNT}, >)$. For many such aggregate test pairs (α, θ), we show a *trichotomy result*: For HAVING queries using (α, θ) without self joins over tuple-independent probabilistic databases, exactly one of the following three statements is true: (1) The exact evaluation problem has P-time data complexity. In this case we call the query *safe*. (2) The exact evaluation problem is ♯P-hard, but the approximate evaluation problem has (randomized) P-time data complexity (there exists an FPTRAS to evaluate the query). In this case, we call the query *apx-safe*. (3) The exact evaluation problem is ♯P-hard and the approximate evaluation problem is also hard (no FPTRAS exists). We call these queries *hazardous*. It is interesting to note that the third class is empty for EXISTS, which are essentially the class of Boolean conjunctive queries that are studied by prior work [46]; that is, all Boolean conjunctive queries have an efficient approximation algorithm.

The approximation algorithms in this chapter are Monte-Carlo-style approximation algorithms; the key technical step such algorithms must perform is efficient sampling, i.e., randomly generating instances (called possible worlds). Computing a random possible world is straightforward in a probabilistic database: we select each tuple with its corresponding marginal probability taking care never to select two disjoint tuples. However, to support efficient techniques like *importance sampling* [102], we need to do something more: we need to generate a random possible world from the set of worlds that *satisfy a constraint that is specified by an aggregate query*. For example, we need to generate a random world, $\tilde{W}$, such that the MAX price returned by a query q on $\tilde{W}$ is equal to 30. We call this the *random possible world generation problem*. Our key technical result is that when q is safe (without aggregation) and the number of elements in the semiring S is small, then we can solve this problem efficiently, i.e., with randomized polynomial time data complexity. The novel technical insight is that *we can use safe plans as a guide to sample the database*. This use is in contrast to the traditional use for safe plans of computing query probabilities exactly. We apply

our novel sampling technique to provide an FPTRAS to approximately evaluate some HAVING queries that have ♯P-hard exact complexity. Thus, the approaches described in this chapter can efficiently answer strictly more queries than our previous, exact approach (albeit only in an approximate sense).

Contributions and Outline

We study conjunctive queries with HAVING predicates on common representations of probabilistic databases [13, 137, 168] where the aggregation function is one of EXISTS, MIN, MAX, COUNT, SUM, AVG, or COUNT(DISTINCT); and the aggregate test is one of $=, \neq, <, \leq, >$, or $\geq$. In Section 4.2, we formalize HAVING queries, our choice of representation, and define efficient evaluation. In Section 4.3, we review the relevant technical background (e.g., semirings and safe plans). In Section 4.4, we give our main results for exact computation: For each aggregate α, we find a class of HAVING queries, called α-safe, such that for any Q using α:

- If Q is α-safe then Q's data complexity is in P.
- If Q has no self joins and is not α-safe then, Q has ♯P-hard data complexity.
- We can decide in polynomial time (in the size of Q) if Q is α-safe.

In Section 4.5, we state and solve the problem of generating a random possible world when the query defining the constraint is safe. In Section 4.6, we discuss approximation schemes for queries that have $\alpha \in \{$MIN,MAX, COUNT, SUM$\}$. The hardness of an approximation algorithm for a HAVING query depends on the aggregate, α, but also on the predicate test, θ. We show:

- If Q is (α, θ)-apx-safe then Q has an FPTRAS.
- If Q has no self joins and is not (α, θ)-apx-safe then, Q does not have an FPTRAS and is (α, θ)-hazardous.
- We can decide in polynomial time (in the size of Q) if Q is (α, θ)-apx-safe.

```
SELECT m.Title
FROM MovieMatch m, Reviewer r
WHERE m.ReviewTitle = r.ReviewTitle
GROUP BY m.Title
HAVING COUNT(DISTINCT r.reviewer) ≥ 2
```

(a) SQL Query

$Q(m)[\texttt{COUNT(DISTINCT } r) \geq 2] \text{ :- } \texttt{MovieMatch}(t, m), \texttt{Reviewer}(-, r, t)$

(b) Extended Syntax (Not Boolean)

$Q[\texttt{COUNT(DISTINCT } r) \geq 2] \text{ :- } \texttt{MovieMatch}(t, \text{`Fletch'}), \texttt{Reviewer}(-, r, t)$

(c) Syntax of this paper

Figure 4.2: A translation of the query *"Which movies have been reviewed by at least 2 distinct reviewers?"* into (a) SQL; (b) an extended syntax of this paper, which is not Boolean; and (C) the syntax of this paper, which is Boolean and is a `HAVING` query.

We show that the trichotomy holds for all combinations of α and $\theta \in \{=, \leq, <, \geq, >\}$, but leave open the case of `COUNT` and `SUM` with either of $\{\geq, >\}$. Additionally, we also show that queries with self joins belong to a complexity class that is believed to be as hard to approximate as any problem in $\sharp\mathsf{P}$. This suggests that the complexity for `HAVING` query approximation is perhaps more subtle than for Boolean queries.

4.2 Formal Problem Description

We first define the syntax and semantics of `HAVING` queries on probabilistic databases and then define the problem of evaluating `HAVING` queries.

4.2.1 Semantics

We consider the aggregate functions `EXISTS`, `MIN`, `MAX`, `COUNT`, `SUM`, `AVG`, and `COUNT(DISTINCT)` as functions on multisets with the obvious semantics.

Definition 4.2.1. *A Boolean conjunctive query is a single rule $q = g_1, \ldots, g_m$ where for $i = 1, \ldots, m$, g_i is a distinct, positive extensional database predicate (EDB), that is, a relational symbol*[5]*. A Boolean `HAVING` query is a single rule:*

$$Q[\alpha(y)\ \theta\ k] \text{ :- } g_1, \ldots, g_n$$

[5]Since all relational symbols are distinct, `HAVING` queries do not contain self joins: $q = R(x, y), R(y, z)$ has a self-join, while $R(x, y), S(y)$ does not.

where for each i, g_i is a positive EDB predicate, $\alpha \in \{$`MIN`, `MAX`, `COUNT`, `SUM`, `AVG`, `COUNT(DISTINCT)`*$\}$, y is a single variable*[6]*, $\theta \in \{=, \neq, <, \leq, >, \geq\}$, and k is a constant. The set of variables in the body of Q is denoted* $\mathbf{var}(Q)$*. We assume that $y \in \mathbf{var}(Q)$. The conjunctive query $q = g_1, \ldots, g_n$, is called the* skeleton *of Q and is denoted* $\mathbf{sk}(Q) = q$*. In the above syntax, θ is called the* predicate test*; k is called the* predicate operand*; and the pair (α, θ) is called an* aggregate test*.*

Figure 4.2(a) shows a SQL query with a `HAVING` predicate that asks for all movies reviewed by at least two distinct reviewers. A translation of this query into an extension of our syntax is shown in Figure 4.2(b). The translated query is not a Boolean `HAVING` query because it has a head variable (m). In this paper, we discuss only Boolean `HAVING` queries. As is standard, to study the complexity of non-Boolean queries, we can substitute constants for head variables. For example, if we substitute 'Fletch' for m, then the result is Figure 4.2(c) which is a Boolean `HAVING` query.

Definition 4.2.2. *Given a* `HAVING` *query $Q[\alpha(y)\ \theta\ k]$ and a world W (a standard relational instance), we define $\mathcal{Y}$ to be the multiset of values $v(y)$ where y is distinguished variable in Q and v is a valuation of $q = \mathbf{sk}(Q)$ that is contained in W. In symbols,*

$$\mathcal{Y} = \{|\ v(y) \mid v \text{ is a valuation for } \mathbf{sk}(Q) \text{ and } \mathbf{im}(v) \subseteq W\ |\}$$

Here, $\mathbf{im}(v) \subseteq W$ denotes that image of $\mathbf{sk}(Q)$ under the valuation v is contained in the world W. We say that Q is satisfied on W and write $W \models Q[\alpha(y)\ \theta\ k]$ (or simply $W \models Q$) if $\mathcal{Y} \neq \emptyset$ and $\alpha(\mathcal{Y})\ \theta\ k$ holds.

In the above definition, we follow SQL semantics and require that $\mathcal{Y} \neq \emptyset$ in order to say that $W \models Q$. For example, $Q[\texttt{COUNT}(*) < 10] :- R(x)$ is false in SQL if $R^W = \emptyset$, i.e., the interpretation of R in the world W is the empty table. This, however, is a minor technicality and our results are unaffected by the alternate choice that $\texttt{COUNT}(*) < 10$ is true on the empty database.

4.2.2 Probabilistic Databases

The data in Figure 4.3 shows an example of a BID database that stores data from integrating extracted movie reviews from USENET with a movie database from IMDB. The `MovieMatch` table

[6]For `COUNT`, we will omit y and write the more familiar $\texttt{COUNT}(*)$ instead.

Title	Matched	P	
‘Fletch’	‘Fletch’	0.95	m_1
‘Fletch’	‘Fletch 2’	0.9	m_2
‘Fletch 2’	‘Fletch’	0.4	m_3
‘The G. Child’	‘The G. Child’	0.95	m_4
‘The G. Child’	‘G. Child’	0.8	m_5
‘The G. Child’	‘Wild Child’	0.2	m_6

`MovieMatch`(CleanTitle, ReviewTitle ;; P)

RID	Reviewer	Title	P	
231	‘Ryan’	‘Fletch’	0.7	t_{231a}
		‘Spies Like Us’	0.3	t_{231b}
232	‘Ryan’	‘Euro. Vacation’	0.90	t_{232a}
		‘Fletch 2’	0.05	t_{232b}
235	‘Ben’	‘Fletch’	0.8	t_{235a}
		‘Wild Child’	0.2	t_{235b}

`Reviews`(RID, Reviewer ; ReviewTitle ; P)

Figure 4.3: Sample data arising from integrating automatically extracted reviews from a movie database. `MovieMatch` is a probabilistic relation, we are uncertain which review title matches with which movie in our clean database. `Reviews` is uncertain because it is the result of *information extraction.*

is uncertain because it is the result of an automatic matching procedure (or fuzzy-join [31]). For example, the probability a review title ‘Fletch’ matches a movie titled ‘Fletch’ is very high (0.95), but it is not certain (1.0) because the title is extracted from text and so may contain errors. For example, from *‘The second Fletch movie’*, our extractor will likely extract just *‘Fletch’* although this review actually refers to *‘Fletch 2’*. The review table is uncertain because it is the result of *information extraction.* That is, we have extracted the title from text (e.g., *‘Fletch is a great movie, just like Spies Like Us’*). Notice that $t_{232a}[P] + t_{232b}[P] = 0.95 < 1$, which indicates that there is some probability reviewid 232 is actually not a review at all.

Query Semantics Users write queries on the possible worlds schema, i.e., their queries do not explicitly mention the probability attributes of relations. In this paper, all queries are Boolean so the answer to a query is a probability score (the marginal probability that the query is true). We define this formally:

Definition 4.2.3 (Query Semantics)**.** *The marginal probability of a `HAVING` query Q on BID database J is denoted $\mu_J(Q)$ (or simply $\mu(Q)$) and is defined by:*

$$\mu_J(Q) = \sum_{W \in \mathcal{W}_J : W \models Q} \mu_J(W)$$

In general, for a Boolean conjunctive query q, we write $\mu_J(q)$ to denote the marginal probability that q is true.

Example 4.2.4 Figure 4.2(c) shows a query that asks for all movies that were reviewed by at least 2 different reviewers. The movie 'Fletch' is present when the following formula is satisfied: $(m_1 \wedge t_{231a}) \vee (m_2 \wedge t_{232b}) \vee (m_1 \wedge t_{235a})$. The multiplicity of tuples returned by the query is exactly the number of disjuncts satisfied. Thus, $\mu(Q)$ is the probability that at least two of these disjuncts are true. Definition 4.2.3 tells us that, semantically, we can compute this by summing over all possible worlds.

4.2.3 Notions of complexity for `HAVING` queries

In the database tradition, we would like to measure the data complexity [164], i.e., treat the query as fixed, but allow the data to grow. This assumption makes sense in practice because the query is generally orders of magnitude smaller than the size of the database. Hence, a running time for query evaluation of $O(n^{f(|Q|)})$ where $|Q|$ is the size of a conjunctive query Q is P-time. In our setting, this introduces a minor technical problem: By fixing a `HAVING` query q, we also fix k (the predicate operand); this means that we should accept a running time $n^{f(k)}$ as efficient. Clearly this is undesirable: because k can be large[7]. For example, $Q[\texttt{SUM}(y) > 200] :- R(x, y)$. For that reason, we consider in this paper an alternative definition of the data complexity of `HAVING` queries, where both the database and k are part of the input.

Definition 4.2.5. *Fix a skeleton q, an aggregate α, and a comparison operator θ. The* query evaluation problem *is: given as input a BID representation J* and *a parameter $k > 0$, calculate $\mu_J(Q)$ where $Q[\alpha(y)\ \theta\ k]$ is such that* $\mathbf{sk}(Q) = q$.

The technical problem that we address in this work is the complexity of the query evaluation problem. Later, we will see that the query evaluation problem for the query in Example 4.2.4 is hard for $\sharp\mathsf{P}$, and moreover, that this is the general complexity for all `HAVING` queries.

---

[7] If we fix the query than k is assumed to be a constant, and so we can take even double exponential time in k. Thus, we would like to take k as part of the input.

4.3 Preliminaries

We review some basic facts about semirings (for a reference see Lang [109]). Then, we introduce random variables over semirings.

4.3.1 Background: Queries on databases with semiring annotations

In this section, we review material from Green *et al.* [81] that tells us how to compute queries on a database whose tuples are annotated with elements of a semiring. To get there, we need some classical definitions.

A *monoid* is a triple $(S, +, 0)$ where S is a set, $+$ is an associative binary operation on S, and 0 is the identity of $+$, i.e., $s + 0 = 0$ for each $s \in S$. For example, $S = \mathbb{N}$ (the natural numbers) with addition is the canonical example of a monoid.

A *semiring* is a structure $(S, +, \cdot, 0, 1)$ where $(S, +, 0)$ forms a commutative monoid with identity 0; $(S, \cdot, 1)$ is a monoid with identity 1; $\cdot$ distributes over $+$, i.e., $s \cdot (t + u) = (s \cdot t) + (s \cdot u)$ where $s, t, u \in S$; and 0 annihilates S, i.e., $0 \cdot s = 0$ for any $s \in S$.

A *commutative semiring* is one in which $(S, \cdot, 1)$ is a commutative monoid. As is standard, we abbreviate either structure with the set S when the associated operations and distinguished constants are clear from the context. In this paper, all semirings will be commutative semirings.

Example 4.3.1 [Examples of Semirings] For an integer $k \geq 0$, let $\mathbb{Z}_{k+1} = \{0, 1, \ldots, k\}$ then for every such k, $(\mathbb{Z}_k, \max, \min, 0, k)$ is a semiring. In particular, $k = 2$ is the Boolean semiring, denoted $\mathbb{B}$. For $k = 1, 2, \ldots,$, another set of semirings we consider are $\mathbb{S}_k = (\mathbb{Z}_k, +_k, \cdot_k, 0, 1)$ where $+_k(x, y) = \min(x + y, k)$ and $\cdot_k = \min(xy, k)$ where addition and multiplication are in $\mathbb{Z}$.

The idea is that database elements will be annotated with elements from the semiring (defined next) and then these annotations will be propagated during query processing. For us, the important point is that aggregation queries can be viewed as doing computation in these semirings.

Definition 4.3.2. *Given a commutative semiring S and a Boolean conjunctive query $q = g_1, \ldots, g_n$, an annotation is a set of functions indexed by subgoals such that for $i = 1, \ldots, n$, τ_{g_i} is a function from tuples that unify with g_i to S. We denote the set of annotation functions with τ.*

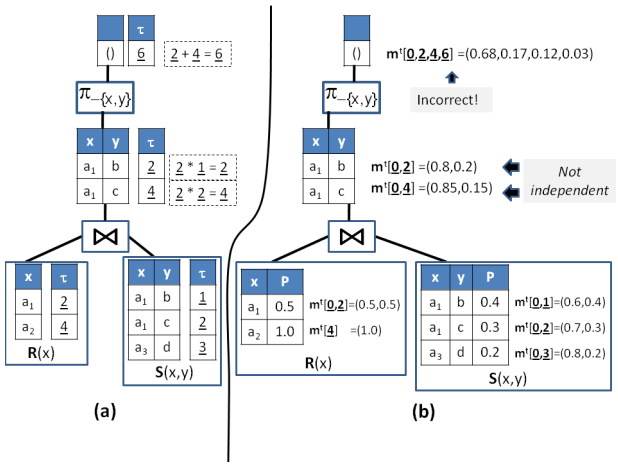


Figure 4.4: (a) This is a query plan $P = \pi_{-x}(\pi_{-y}(R(x) \bowtie S(x,y)))$ for the query $q = R(x), S(x,y)$ over some database annotated in $\mathbb{N}$. The value of the query is $q(W,\tau) = 6$. (b) This is an extensional plan (Definition 4.4.5) for P ($\pi^I_{-x}(\pi^I_{-y}(R(x) \bowtie S(x,y)))$). This plan is not safe, since intermediate values may be neither independent nor disjoint. Thus, the extensional value computed by this plan is not the correct marginal probability of the query. For readability, we underline elements of the semiring.

Remark 4.3.3. *In the above definition, we restrict τ to assigning values to tuples that unify with g_i, since g_i may incorporate selections. For example, if $g_i = R(x, \text{'}a\text{'})$ then τ does not need to assign values to tuples whose second component is 'b'. Implicitly, τ should assign all such tuples* 0.

We recall the syntax of relational plans as we generalize them in this work.

Definition 4.3.4 (Query Plan Syntax)**.**

- *a* plan P *is inductively defined as (1) a single subgoal that may include selections, (2) $\pi_{-x}P_1$ if P_1 is a plan and x is a variable, and (3) $P_1 \bowtie P_2$ if P_1, P_2 are plans.*

- $\mathbf{var}(P)$*, the variables output by P, is defined inductively as (1)* $\mathbf{var}(g)$*, the variables in the subgoal g, if $P = g$; (2)* $\mathbf{var}(\pi_{-x}P) = \mathbf{var}(P) - \{x\}$*; and (3)* $\mathbf{var}(P_1 \bowtie P_2) = \mathbf{var}(P_1) \cup \mathbf{var}(P_2)$.

- $\mathbf{goal}(P)$*, the set of subgoals in P, is defined inductively as (1)* $\mathbf{goal}(g) = \{g\}$*; (2)* $\mathbf{goal}(\pi_{-x}P_1) = \mathbf{goal}(P_1)$*; and (3)* $\mathbf{goal}(P_1 \bowtie P_2) = \mathbf{goal}(P_1) \cup \mathbf{goal}(P_2)$.

A graphical example query plan is shown in Figure 4.4(a) along with its description in the above syntax.

We view relational plans as computing relational tuples that are annotated with elements of a semiring (following Green *et al.* [81]). To be precise, fix a domain $\mathbb{D}$, and denote the *value* of a plan P on a deterministic instance W as ω_P^W, which is a function $\mathbb{D}^{|\mathbf{var}(P)|} \to S$. Informally, the value of a plan maps each standard tuple returned by the plan to an element of the semiring S. We define ω_P^W inductively:

- If $P = g$ then if $t \in W$ and t unifies with g then $\omega_P^W(t) = \tau_g(t)$ else $\omega_P^W(t) = 0$.
- If $P = \pi_{-x} P_1$, then $\omega^W_{\pi_{-x}P_1}(t) = \sum_{t' : t'[\mathbf{var}(P)]=t} \omega^W_{P_1}(t')$.
- else $P = P_1 \bowtie P_2$ and for $i = 1, 2$ let t_i be t restricted to $\mathbf{var}(P_i)$ then $\omega^W_{P_1 \bowtie P_2}(t) = \omega^W_{P_1}(t_1) \cdot \omega^W_{P_2}(t_2)$

An example of a plan computing a value in a semiring is shown in Figure 4.4(a). The value of the plan in the figure is 6: Since the plan is Boolean, it returns the empty tuple which is annotated with 6, more succinctly, $\omega_P^W() = 6$.

For a standard conjunctive query q, there may be many distinct, but logically equivalent, relational plans to compute q. Green *et al.* [81] show that ω_P^W *does not depend* on the particular choice of logically equivalent plan P for q. In turn, this justifies the notation $q(W, \tau)$, as the value of a conjunctive query q on a deterministic instance W under annotation τ. Formally, we define this value as $q(W, \tau) \stackrel{\text{def}}{=} \omega_P^W()$ where P is any plan for q and where ω_P^W is applied to the empty tuple. This notion is well defined precisely because the value of q does not depend on the choice of plan, P. When τ is clear from the context, we drop it and write simply $q(W)$ to denote the value of q on a world W.

4.3.2 Background: Random Variables on Semirings

In this section, we extend the idea of semirings on a standard database to probabilistic databases. Intuitively, in each possible world, every tuple is annotated with a (potentially different) semiring element. Hence, we think of each tuple as being associated with a *semiring random variable* (defined formally below). A naive representation of these random variables can be large, which motivates us

to define an efficient (small) representation called *marginal vectors* (in full analogy with marginal probabilities). In addition, we define (efficient) operations on these marginal vectors that are fully analogous with multiplying and adding marginal probabilities . In the remainder of this section, we fix a BID instance J, and denote by $(\mathcal{W}, \mu)$ the distribution on possible worlds induced by J (Section 4.2.1).

Definition 4.3.5. *Given a semiring S, an S-*random variable*, r, is a function $r : \mathcal{W} \to S$. Given two S-random variables r, t then $r + t$ and $r \cdot t$ denote random variables defined in the obvious way:*

$$(r + t)(W) = r(W) + t(W) \text{ and } (r \cdot t)(W) = r(W) \cdot t(W)$$

We write $r = s$ as a shorthand for the event that the random variable r takes value s. We denote the probability of this event as $\mu(r = s)$. More precisely, $\mu(r = s) = \mu(\{W \in \mathcal{W} \mid r(W) = s\})$. Two basic notions on random variables are independence and disjointness:

Definition 4.3.6. *Given a semiring S and a set of random variables $R = \{r_1, \ldots, r_n\}$ on S, R is* independent *if $\forall N \subseteq \{1, \ldots, n\}$ and any set $s_1, \ldots, s_n \in S$, we have*

$$\mu\left(\bigwedge_{i \in N} r_i = s_i\right) = \prod_{i \in N} \mu(r_i = s_i)$$

We say that R is disjoint *if for any $i \neq j$ we have:*

$$\mu((r_i \neq 0) \wedge (r_j \neq 0)) = 0$$

If r and t are two disjoint random variables[8] then $\mu(r = 0 \vee t = 0) = \mu(r = 0) + \mu(t = 0) - 1$.

To represent a single S-random variable, we may need space as large as the number of possible worlds ($|\mathcal{W}|$). This can be exponential in the size of the database J, and so, is prohibitive for most applications. We now define an alternative representation called *marginal vectors* that have size proportional to the size of the semiring, i.e., $|S|$.

[8] A more illustrative way to write this computation is $\mu[r = 0 \vee t = 0] = 1 - \mu[r \neq 0 \wedge t \neq 0] = 1 - (1 - \mu(r = 0)) + (1 - \mu(t = 0))$

Definition 4.3.7. *Given a random variable r on S, the* marginal vector *(or simply, the marginal) of r is denoted $\boldsymbol{m}^r$ and is a real-valued vector indexed by S defined by $\forall s \in S \ \mu(r = s) = \boldsymbol{m}^r[s]$.*

Two simple facts immediate from the definition are $\forall s \in S \ m^r[s] \geq 0$ (all entries are positive) and $\sum_{s \in S} m^r[s] = 1$ (total probability). We use the following notation $\boldsymbol{m}^r[s_1, \ldots, s_k]$ where $s_1, \ldots, s_k$ are semiring elements to be a shorthand for the tuple of marginal probabilities $(\boldsymbol{m}^r[s_1], \ldots, \boldsymbol{m}^r[s_k])$.

Marginal vectors for semiring random variables are the analog of marginal probabilities for Boolean events: they are a means to write down a simple, succinct (but lossy) representation of a random variable. In the case of a Boolean semiring (i.e., $\mathbb{B} = (\{0, 1\}, \max, \min, 0, 1)$), a random variable r is an event that is true (when $r = 1$) or false (when $r = 0$). Suppose that the marginal probability that r is true is p_r (and so it is false with probability $1 - p_r$). Then, the marginal vector has two entries one for each of the semiring elements, 0 and 1:

$$\boldsymbol{m}^r[0] = 1 - p_r \text{ and } \boldsymbol{m}^r[1] = p_r$$

If r and t are independent Boolean events, then their conjunction $r \wedge t$ has marginal probability given by the simple formula $\mu[r \wedge t] = \mu[r]\mu[t]$. We generalize the idea of multiplying marginal probabilities to marginal vectors of semiring elements; the resulting operation is called a *monoid convolution*. In full analogy, when when r, t are disjoint semiring random variables, we introduce a *disjoint operation* that is analogous to the rule $\mu[r \vee t] = \mu[r] + \mu[t]$ for disjoint Boolean events.

Definition 4.3.8. *Given a monoid $(S, +, 0)$, the* monoid convolution *is a binary operation on marginal vectors denoted $\oplus$. For any marginals $\boldsymbol{m}^r$ and $\boldsymbol{m}^t$ we define the s-entry (for $s \in S$) of $\boldsymbol{m}^r \oplus \boldsymbol{m}^t$ by the equation:*

$$(\boldsymbol{m}^r \oplus \boldsymbol{m}^t)[s] \stackrel{\text{def}}{=} \sum_{i,j : i+j=s} \boldsymbol{m}^r[i]\boldsymbol{m}^t[j]$$

That is, the sum ranges over all pairs of elements from the semiring S whose sum (computed in the semiring S) is exactly s. We emphasize that since the entries of the marginal vectors are in $\mathbb{R}$, the arithmetic operations on $\boldsymbol{m}$ in the above equation are performed in $\mathbb{R}$ as well.

The disjoint operation *for $(S, 0, +)$ is denoted $\boldsymbol{m}^r \bigsqcup \boldsymbol{m}^t$ and is defined by*

if $s \neq 0$ $(\boldsymbol{m}^r \bigsqcup \boldsymbol{m}^t)[s] \stackrel{\text{def}}{=} \boldsymbol{m}^r[s] + \boldsymbol{m}^t[s]$

else $(\boldsymbol{m}^r \bigsqcup \boldsymbol{m}^t)[0] \stackrel{\text{def}}{=} (\boldsymbol{m}^r[0] + \boldsymbol{m}^t[0]) - 1.$

In a semiring $(S, +, \cdot, 0, 1)$ *we use* $\oplus$ *to mean the convolution over addition, i.e., over the monoid* $(S, +, 0)$*, and* $\otimes$ *to mean the convolution over multiplication, i.e., over the monoid* $(S, \cdot, 1)$*. Notice that the disjoint operation is always paired with* $+$ *(not* $\cdot$*).*

Example 4.3.9 Consider the Boolean semiring $\mathbb{B}$ and two random variables r and t taking values in $\mathbb{B}$ with marginal probabilities p_r and p_t, respectively. Then $\boldsymbol{m}^r = (1-p_r, p_r)$ and $\boldsymbol{m}^t = (1-p_t, p_t)$. If r and t are independent, then the distribution of $r \vee t$ can be computed using $r \oplus t$ (in $\mathbb{B}$, $r \vee t = r + t$). From the definition, we see that $(r \oplus t)[0] = (1-p_r)(1-p_t)$ and $(r \oplus t)[1] = (1-p_t) + (1-p_r)p_t + p_r p_t$.

If r and t are disjoint, then $\boldsymbol{m}^{r+t}[1] = (\boldsymbol{m}^r \coprod \boldsymbol{m}^t)[1] = (p_r + p_t)$ and $\boldsymbol{m}^{r+t}[0] = (\boldsymbol{m}^r \coprod \boldsymbol{m}^t)[0] = 1 - \boldsymbol{m}^{r+t}[1]$.

The next proposition restates that the two operations in the previous definition yield the correct results, and states bounds on their running time:

Proposition 4.3.10. *Let* r *and* s *be random variables on the monoid* $(S, +, 0)$ *with marginal vectors* $\boldsymbol{m}^r$ *and* $\boldsymbol{m}^t$*, respectively. Then let* $\boldsymbol{m}^{r+t}$ *denote the marginal of* $r + t$*. If* r *and* t *are independent then* $\boldsymbol{m}^{r+t} = \boldsymbol{m}^r \oplus \boldsymbol{m}^t$*. If* r *and* t *are disjoint then* $\boldsymbol{m}^{r+t} = \boldsymbol{m}^r \coprod \boldsymbol{m}^t$*. Further, the convolution is associative, so the convolution of* n *variables* $r_1, \ldots, r_n$ *can be computed in time* $O(n\,|S|^2)$*:*

$$\bigoplus_{i=1,\ldots,n} \boldsymbol{m}^{r_i} \stackrel{\text{def}}{=} \boldsymbol{m}^{r_1} \oplus \cdots \oplus \boldsymbol{m}^{r_n}$$

and disjoint operation applied to $r_1, \ldots, r_n$ *denoted below can be computed in* $O(n\,|S|)$*.*

$$\coprod_{i=1,\ldots,n} \boldsymbol{m}^{r_1} \stackrel{\text{def}}{=} \boldsymbol{m}^{r_1} \coprod \cdots \coprod \boldsymbol{m}^{r_n}$$

Proof. We include the proof of the convolution since it is illustrative. We assume that $\boldsymbol{m}^x[i] = \mu(x = i)$ for $x \in \{r, t\}$ and $i \in S$, i.e., the marginal vectors are correct, and that r and t are independent. We

show that $(\boldsymbol{m}^r \oplus \boldsymbol{m}^t)[s] = \mu(r + t = s)$. Since $s \in S$ is arbitrary, this proves the correctness claim.

$$\begin{aligned}
\left(\boldsymbol{m}^r \oplus \boldsymbol{m}^t\right)[s] &= \sum_{i,j \in S : i+j=s} \boldsymbol{m}^r[i]\boldsymbol{m}^t[j] \\
&= \sum_{i,j \in S : i+j=s} \mu(r = i)\mu(t = j) \\
&= \sum_{i,j \in S : i+j=s} \mu(r = i \wedge t = j) \\
&= \mu(r + t = s) = \boldsymbol{m}^{r+t}[s]
\end{aligned}$$

The first equality is the definition. The second equality is by assumption that the marginal vectors are correct. The third line is by the independence assumption. The final line is because the sum is exhaustive. To see the time bound, observe that we can simply consider all $|S|^2$ pairs to compute the convolution (which we assume has unit cost). Since the semiring is associative, and so is the convolution. This also means that we can compute the n-fold convolutions pairwise. □

The importance of this proposition is that if the number of elements in the semiring is small, then each operation can be done efficiently. We will use this proposition as the basis of our efficient exact algorithms.

4.4 Approaches for `HAVING`

We define α-safe `HAVING` queries for $\alpha \in \{$ `EXISTS`, `MIN`, `MAX`, `COUNT`$\}$ in Section 4.4.3, for $\alpha =$ `COUNT(DISTINCT)` in Section 4.4.4, and $\alpha \in \{$`AVG`, `SUM`$\}$ in Section 4.4.5.

4.4.1 Aggregates and semirings

We explain how to compute `HAVING` queries using semirings on deterministic databases, which we then generalize to probabilistic databases. Since `HAVING` queries are Boolean, we use a function $\rho : S \to \{\text{true}, \text{false}\}$, called the *recovery function*, that maps a semiring value s to true if that value satisfies the predicate in the having query Q, e.g., when checking `COUNT`$(*) \geq 4$, $\rho(4)$ is true, but $\rho(3)$ is false. Figure 4.5 lists the semirings for the aggregates in this paper, their associated

HAVING Predicate	Semiring	Annotation $\tau_{g^*}(t)$	Recovery $\rho(s)$
EXISTS	$(\mathbb{Z}_2, \max, \min)$	1	$s = 1$
MIN(y) $\{<, \leq\}$ k	$(\mathbb{Z}_3, \max, \min)$	if $t\ \theta\ k$ then 2 else 1	$s = 2$
MIN(y) $\{>, \geq\}$ k	$(\mathbb{Z}_3, \max, \min)$	if $t\ \theta\ k$ then 1 else 2	$s = 1$
MIN(y) $\{=, \neq\}$ k	$(\mathbb{Z}_4, \max, \min)$	if $t < k$ then 3 else if $t = k$ then 2 else 1	if = then $s = 2$ if $\neq$ then $s \neq 2$
COUNT($*$) θ k	$\mathbb{S}_{k+1}$	1	$(s \neq 0) \wedge (s\ \theta\ k)$
SUM(y) θ k	$\mathbb{S}_{k+1}$	$t[y]$	$(s \neq 0) \wedge (s\ \theta\ k)$

Figure 4.5: Semirings for the operators MIN, COUNT and SUM. Let g^* be the lowest indexed subgoal such that contains y. For all $g \neq g^*$, $\forall t$, $\tau_g(t)$ equals the multiplicative identity of the semiring. Let $\mathbb{Z}_{k+1} = \{0, 1, \ldots, k\}$ and $+_k(x, y) \stackrel{\text{def}}{=} \min(x + y, k)$ and $\cdot_k \stackrel{\text{def}}{=} \min(xy, k)$, where $x, y \in \mathbb{Z}$. Let $\mathbb{S}_k \stackrel{\text{def}}{=} (Z_{k+1}, +_k, \cdot_k, 0, 1)$. MAX and MIN are symmetric. COUNT(DISTINCT) is omitted because it uses two different algebras together. One important point to note is that, in the case of SUM, if t is outside the semiring (i.e., larger) than $\tau(t)$ is set to the largest element of the semiring. Since all values are present, once this value is present it forces the value of the predicate θ, e.g., if $\theta =\geq$ then the predicate is trivially satisfied.

annotation functions τ, and an associated Boolean recovery function ρ. The aggregation function EXISTS essentially yields the safe plan algebra of Dalvi and Suciu [46, 48, 135].

Example 4.4.1 Consider the query $Q[\text{MIN}(y) > 10] :- R(y)$ where $R = \{t_1, \ldots, t_n\}$ is a tuple independent database. Figure 4.5 tells us that we should use the semiring $(\mathbb{Z}_3, \max, \min)$. We first apply τ: $\tau(t_i) = 1$ represents that $t_i[y] > 10$ while $\tau(t_i) = 2$ represents that $t_i[y] \leq 10$. Let $q_\tau = \sum_{i=1,\ldots,m} \tau(t_i)$, the sum is in S, and so, $q_\tau = \max_{i=1,\ldots,m} \tau(t_i)$. Now, $\rho(q_t)$ is satisfied only when q_τ is 1. In turn, this occurs if and only if all $t_i[y]$ are greater than 10 as required.

A careful reader may have noticed that we could have used $\mathbb{Z}_2$ to compute this example (instead of $\mathbb{Z}_3$). When we generalize to probabilistic databases, we may have to account for a tuple being absent (for which we use the value 0).

More generally, we have the following proposition:

Proposition 4.4.2. *Given a HAVING query Q, let $q = \mathbf{sk}(Q)$ and S, ρ and τ be chosen as in Fig-*

ure 4.5, then for any deterministic instance W:

$$W \models Q \iff \rho\left(q(W, \tau)\right)$$

Proof. Let q, the skeleton of Q, have n subgoals. We show only `MIN` with $\leq$ in full detail. All other aggregate-test pairs follow by similar arguments. We observe the equation

$$q(W, \tau) = \sum_{v:\mathbf{im}(v)\subseteq W} \prod_{i=1,\ldots,n} v(g_i)$$

Further, $W \models Q[\texttt{MIN}(y) \leq k]$ if and only if there there is some valuation such that $\prod_{i=1,\ldots,n} v(g_i) = 2$. Since, $2 + s = 2$ for any $s \in S$ the existence of such a valuation implies $q(W, \tau) = 2$. Conversely, if $q(W, \tau) = 2$ then there must be some such valuation since $x + y = 2$ implies that either x or y is 2 in this semiring. Hence, the claim holds.

Similarly, $W \models Q[\texttt{MIN}(y) \geq k]$, the query is satisfied if and only if *all* elements are $\geq k$ and so each term (valuation) in the summation must evaluate to 0 or 1. Similar arguments are true for $=, \neq$. In the case of `COUNT`, if we want to count from $1, \ldots, k$ we also need two elements, 0 and $k + 1$: 0 encodes that a tuple is absent and $k + 1$ encodes that the value is "bigger than k". □

In probabilistic databases, we view $q(W, \tau)$ as a random variable by fixing τ (the semiring annotation functions), i.e., we view $q(W, \tau)$ as a function of W alone. We denote this random variable q_τ. Our goal is to compute the marginal vector of q_τ. The marginal vector of q_τ, denoted $\boldsymbol{m}^{q_\tau}$, is sufficient to compute the value of any `HAVING` query since we can simply examine those entries in $\boldsymbol{m}^{q_\tau}$ for which the recovery function, ρ, is true. Said another way, a simple corollary of Prop. 4.4.2 is the following generalization to probabilistic databases:

Corollary 4.4.3. *Given a `HAVING` query Q, let $q = \mathbf{sk}(Q)$, S, ρ, and τ be as in Prop. 4.4.2, then for any BID instance J we have the following equalities:*

$$\mu_J(Q) = \sum_{k \,:\, \rho(k) \text{ is true}} \boldsymbol{m}^{q_\tau}[k]$$

Cor. 4.4.3 tells us that we can compute $\mu(Q)$ by examining the entries of the marginal vector $\boldsymbol{m}^{q_\tau}$. Hence, our goal is to compute $\boldsymbol{m}^{q_\tau}[s]$ for each such index, $s \in S$.

4.4.2 Computing safely in semirings

We now extend safe plans to compute a marginal vector instead of a Boolean value. Specifically, we compute $\boldsymbol{m}^{q_\tau}$, the marginal vector for q_τ using the operations defined in Section 4.3.2.

Definition 4.4.4. *An* extensional plan *for a Boolean conjunctive query q is defined recursively as a subgoal g and if P_1, P_2 are extensional plans then so are $\pi^I_{-x}P_1$ (independent project), $\pi^D_{-x}P_1$ (disjoint project), and $P_1 \bowtie P_2$ (join). An extensional plan P is* safe *if, assuming P_1 and P_2 are safe, the following conditions are met:*

- *$P = g$ is always safe*
- *$P = \pi^I_{-x}P_1$ is safe if $x \in \mathbf{var}(P_1)$ and $\forall g \in \mathbf{goal}(P_1)$ then $x \in \mathbf{key}(g)$*
- *$P = \pi^D_{-x}P_1$ is safe if $x \in \mathbf{var}(P_1)$ and $\exists g \in \mathbf{goal}(P_1)$, $\mathbf{key}(g) \subseteq \mathbf{var}(P)$, $x \in \mathbf{var}(g)$.*
- *$P = P_1 \bowtie P_2$ is safe if $\mathbf{goal}(P_1) \cap \mathbf{goal}(P_2) = \emptyset$ and for $i = 1, 2$, $\mathbf{var}(\mathbf{goal}(P_1)) \cap \mathbf{var}(\mathbf{goal}(P_2)) \subseteq \mathbf{var}(P_i)$, i.e., we may not project away variables that are shared in two subgoals before they are joined.*

An extensional plan P is a safe plan for q if P is safe and $\mathbf{goal}(P) = q$ and $\mathbf{var}(P) = \emptyset$.

Intuitively, a safe plan tells us that the correlations of tuples produced by intermediate stages of the plan are either independent or disjoint, as opposed to correlated in some unknown way. In particular, $P = \pi^I_{-x}(P_1)$ is a safe plan whenever those tuples produced by P_1 on any instance are independent (provided the tuples differ on the variable x). Hence, we call π^I an independent project. Similarly, if $P = \pi^D_{-x}(P_1)$ is safe, then the tuples produced by P_1 are disjoint whenever they differ on the variable x. Further, a join is safe if the branches do not contain any common subgoals, i.e., any tuple produced by P_1 is independent of any tuple produced by P_2. For completeness, we state and prove a formal version of this discussion in Appendix B.1.

Computing With Safe Plans

We now augment safe plans to compute marginal vectors. Intuitively, we generalize the operation of multiplying marginal probabilities (as done in safe plans) to semiring convolutions of marginal vectors, and we generalize the operation of adding the marginal probabilities of disjoint events to disjoint operations on marginal vectors. We think of a plan as computing a marginal vector: The marginal vector computed by a plan P on a BID instance J is called the *extensional value* of P and is denoted as $\hat{\omega}^J_{P,S}$ and is defined below.

Definition 4.4.5. *Given a BID instance J and a semiring S. Let P be a safe plan. Denote the* extensional value *of P in S on J as $\hat{\omega}^J_{P,S}$. $\hat{\omega}^J_{P,S}$ is a function that maps each tuple to a marginal vector. To emphasize the recursion, we fix J and S and denote $\hat{\omega}^J_{P,S}$ as $\hat{\omega}_P$. We define the value of $\hat{\omega}_P$ inductively:*

- *If $P = g$ then $\hat{\omega}_P(t) = \boldsymbol{m}^t$ where $\boldsymbol{m}^t[0] = 1 - t[P]$ and $\boldsymbol{m}^t[\tau_g(t)] = t[P]$ and all other entries are 0.*

- *If $P = \pi^I_{-x} P_1$ then $\hat{\omega}_P(t) = \bigoplus_{t' : t'[\mathbf{var}(P_1)] = t} \hat{\omega}_{P_1}(t)$ where $\bigoplus$ denotes the convolution over the monoid $(S, +, 0)$.*

- *If $P = \pi^D_{-x} P_1$ then $\hat{\omega}_P(t) = \coprod_{t' : t'[\mathbf{var}(P_1)] = t} \hat{\omega}_{P_1}(t)$ where $\coprod$ denotes the disjoint operation over the monoid $(S, +, 0)$.*

- *If $P = P_1 \bowtie P_2$ then $\hat{\omega}_P(t) = \hat{\omega}_{P_1}(t_1) \otimes \hat{\omega}_{P_2}(t_2)$ where for $i = 1, 2$ t_i is t restricted to $\mathbf{var}(P_i)$ and $\otimes$ denotes the convolution over the monoid $(S, \cdot, 1)$.*

Figure 4.4(b) gives an example of computing the extensional value of a plan: The plan shown is not safe, meaning that the extensional value it computes is not correct, i.e., equal to $\boldsymbol{m}^{q_\tau}$. This illustrates that any plan may be converted to an extensional plan, but we need additional conditions

(safety) to ensure that the computation is correct. Interestingly, in this case, there is an alternate safe plan: $P_0 = \pi_{-x}(R(x) \bowtie \pi_{-y}(S(x,y)))$, i.e., we move the projection early.

The next lemma states that for safe plans, the extensional value is computed correctly, i.e., the conditions insured by the safe plan and the operator used in Definition 4.4.5 make exactly the same correlation assumptions. For example, π^I indicates independence, which ensures that $\oplus$ correctly combines two input marginal vectors. The proof of the following lemma is a straightforward induction and is omitted.

Lemma 4.4.6. *If P is a safe plan for a Boolean query q and τ is any annotation function into S, then for any $s_i \in S$ on any BID instance J, we have $\hat{\omega}_P^J()[s_i] = \mu_J(q_\tau = s_i)$.*

A safe plan (in the terminology of this paper) ensures that the convolutions and disjoint operations output the correct results, but it is *not sufficient to ensure that the plan is efficient.* In particular, the operations in a safe plan on S take time (and space) polynomial in $|S|$. Thus, if the size of S grows super-polynomially in $|J|$, the size of the BID instance, the plan *will not be efficient.* As we will see, this rapid growth happens for `SUM` in most cases. In contrast, as we show in the next section, if α is one of `MIN`, `MAX`, or `COUNT`, the number of elements in the needed semiring is small enough, so the safety of **sk**(Q) and Q coincide.

4.4.3 EXISTS-,MIN-, MAX- and COUNT-safe

We now give optimal algorithms when α is one of `EXISTS`, `MIN`, `MAX`, or `COUNT`. The results on `EXISTS` are exactly the results of Dalvi and Suciu [46]. We include them to clarify our generalization.

Definition 4.4.7. *Let α be one of {`EXISTS`, `MIN`, `MAX`, `COUNT`} and $Q[\alpha(t)\ \theta\ k]$ be a `HAVING` query, then Q is α-*safe *if the skeleton of Q is safe.*

Theorem 4.4.8. *Let $Q[\alpha(y)\ \theta\ k]$ be a `HAVING` query for $\alpha \in$ { `EXISTS`, `MIN`, `MAX`, `COUNT`} such that Q is α-safe then the exact evaluation problem for Q is in polynomial time in the size of the data.*

Correctness is straightforward from Lemma 4.4.6. Efficiency follows because the semiring is of constant size for `EXISTS`, `MIN`, and `MAX`. For `COUNT`, observe that an upper bound on $|S|$ is number

of tuples returned by the query plus one (for empty), thus count is polynomially bounded as well. Thus, the entire plan has polynomial time data complexity.

Complexity

The results of Dalvi and Suciu [46, 48, 135] show that either a conjunctive query without self joins has a safe plan or it is ♯P-hard. The idea is to show that a `HAVING` query Q is satisfied only if $\mathbf{sk}(Q)$ is satisfied, which implies that computing Q is at least as hard as computing $\mathbf{sk}(Q)$. Formally, we have:

Theorem 4.4.9 (Exact Dichotomy for `MIN`, `MAX`, and `COUNT`)**.** *If $\alpha \in \{\mathit{MIN}, \mathit{MAX}, \mathit{COUNT}\}$ and $Q[\alpha(y)\,\theta\,k]$ does not contain self joins, then either (1) Q is α-safe and so Q has data complexity in* P*, or (2) Q has ♯P-hard data complexity. Further, we can find an α-safe plan in* P.

Proof. The first part of the dichotomy is Theorem 4.4.8. We show the matching negative result. Consider the predicate test $\texttt{MIN}(y) \geq 1$; assuming that Q is not `MIN`-safe, we have (by above) that $\mathbf{sk}(Q) = q$ is not safe in the sense of Dalvi and Suciu, we show that this query can be used to compute $\mu[q]$ on an BID instance J. To see this, create a new instance J' that contains exactly the same tuples as J, but recode all values in attributes referenced by y as integers with values greater than 1: this query is true precisely when at least one tuple exists and hence with $\mu[q]$. We show below that this is sufficient to imply that all tests θ are hard as well. The proof for `MAX` is symmetric. `COUNT` is similar. □

Lemma 4.4.10. *Let $\alpha \in \{\mathit{MIN}, \mathit{MAX}, \mathit{COUNT}, \mathit{SUM}, \mathit{COUNT}(\mathit{DISTINCT})\}$, if computing $Q[\alpha(y) = k]$ exactly is ♯P-hard, then it is ♯P-hard for* all *$\theta \in \Theta$. Furthermore, if q_τ takes at most polynomially many values then the converse also holds: if computing $Q[\alpha(y)\ \theta\ k]$ exactly is ♯P-hard for any $\theta \in \Theta$, then it is ♯P-hard for all $\theta \in \Theta$.*

Proof. We first observe that all aggregate functions α in the statement are positive, integer-valued functions. We show that we can use $\leq, \geq, >, <, \neq$ as a black box to compute = efficiently. We then show that we can compute the inequalities in time $O(k)$ (using =), thus proving both parts of the claim.

First, observe that $q(W, \tau) = s$ is a function on worlds, i.e., the events are disjoint for different values of s. Hence,

$$\mu(Q[\alpha(y) \leq k]) = \sum_{k' \leq k} \mu(Q[\alpha(y) = k']$$

From this equation it follows that we can compute any inequality using = in time proportional to the number of possible values. To see the forward direction, we compute

$$\mu(Q[\alpha(y) \leq k+1]) - \mu(Q[\alpha(y) \leq k]) = \mu(Q[\alpha(y) = k])$$

similarly for a strict inequality. And, $1 - \mu(Q[\alpha(y) \neq k]) - \mu(Q[\alpha(y) \neq 0]) = \mu(Q[\alpha(Y) = k])$. The $\neq 0$ statement is only necessary with SQL semantics. □

The exact ♯P-hardness proofs in the remainder of this section satisfy the requirement of this lemma. Interestingly, this lemma *does not hold for approximation hardness*.

4.4.4 COUNT(DISTINCT)-safe queries

Intuitively, we compute `COUNT(DISTINCT)` in two stages: (1) For the subplan rooted at π_{-y}, we first compute the probability that each value is returned by the plan (i.e., we compute the `DISTINCT`part using `EXISTS`). (2) Then, since we have removed duplicates implicitly using `EXISTS`, we count the number of distinct values using the `COUNT` algorithm from Section 4.4.3.

The ordering of the operators, first `EXISTS` and then `COUNT`, is important. As we show in Theorem 4.4.16, this ordering exactly captures tractable evaluation. First, we need a small technical proposition to state our characterization:

Proposition 4.4.11. *If P is a safe plan for q, then for $x \in \mathbf{var}(q)$ there is exactly one of π^I_{-x} or π^D_{-x} in P.*

Proof. At least one of the two projections must be present, because we must remove the variable x (q is Boolean). If there were more than one in the plan, then they cannot be descendants of each other because $x \notin \mathbf{var}(P_1)$ for the ancestor and they cannot be joined afterward because of the join condition for $i = 1, 2$ $\mathbf{var}(\mathbf{goal}(P_1)) \cap \mathbf{var}(\mathbf{goal}(P_2)) \subseteq \mathbf{var}(P_i)$. □

Thus, it makes sense to talk about the unique node in the plan tree where a variable x is removed, as we do in the next definition:

Definition 4.4.12. *A query $Q[\texttt{COUNT}(\texttt{DISTINCT}\ y)\ \theta\ k]$ is* COUNT(DISTINCT)-safe *if there is a safe plan P for the skeleton of Q such that if P_1 is the unique node in the plan where y is removed, i.e., either π^I_{-y} or π^D_{-y} in P, then no proper ancestor of P_1 is π^I_{-x} for any x.*

This definition exactly insists on the ordering of operators that we highlighted above.

Example 4.4.13 Fix a BID instance J. Consider

$$Q[\texttt{COUNT}(\texttt{DISTINCT}\ y) \geq 2] :- R(y, x), S(y)$$

A COUNT(DISTINCT)-safe plan for the skeleton of Q is $P = \pi^I_{-y}((\pi^I_{-x}R(y, x)) \bowtie S(y))$. The subquery $P_1 = (\pi^I_{-x}R(y, x)) \bowtie S(y)$ returns tuples (values for y). We use the EXISTS algebra to compute the probability that each distinct value appears.

Now, we must count the number of distinct values: Since we have eliminated duplicates, all y values are trivially distinct and we can use the COUNT algebra. To do this, we map each EXISTS marginal vector to a vector suitable for computing COUNT, i.e., a vector in $\mathbb{Z}_k$ (here $k = 2$). In other words, $(1 - p, p) = \hat{\omega}^J_{P,\texttt{EXISTS}}(t) = \boldsymbol{m}^t$ is mapped to $\hat{\tau}(\boldsymbol{m}^t) = (1 - p, p, 0)$. In general, this vector would be of length $k + 1$.

Since $P = \pi^I_{-y}P_1$, we know that all tuples returned by P_1 are independent. Thus, the correct distribution is given by convolution over all such t', each one corresponding to a distinct y value, i.e., $\oplus_t \hat{\tau}(t')$. To compute the final result, use the recovery function, ρ defined by $\rho(s) = s \geq 2$

The proof of the following theorem is a generalization of Example 4.4.13, whose proof we include in the appendix (Appendix B.2):

Theorem 4.4.14. *If Q is COUNT(DISTINCT)-safe then its evaluation problem is* P*-time.*

Complexity We now establish that for COUNT(DISTINCT) queries without self joins, COUNT(DISTINCT)-safe captures efficient computation. We do this in two stages: first, we exhibit some canonical hard patterns for COUNT(DISTINCT), and second, in the appendix, we reduce any other non-COUNT(DISTINCT)-safe pattern to one of these hard patterns.

Proposition 4.4.15. *The following* `HAVING` *queries are* ♯P*-hard for* $i = 1, 2, \ldots$*:*

$$Q_1[\texttt{COUNT}(\texttt{DISTINCT}\, y)\ \theta\ k] \coloneq \mathtt{R}(x), \mathtt{S}(x, y)$$

and,

$$Q_{2,i}[\texttt{COUNT}(\texttt{DISTINCT}\, y)\ \theta\ k] \coloneq \mathtt{R}_1(x; y), \ldots, \mathtt{R}_i(x; y)$$

Proof. We prove Q_1 is hard and defer $Q_{2,i}$ to the Appendix B.2. To see that Q_1 is hard, we reduce from counting the number of independent sets in a graph (V, E) which is ♯P-hard. We let k be the number of edges ($|E|$) and $\theta =$ ' $\geq$'. Intuitively, with these choices Q will be satisfied only when all edges are present. For each node $u \in V$, create a tuple $\mathtt{R}(u)$ with probability 0.5. For edge $e = (u, v)$ create two tuples $\mathtt{S}(u, e), \mathtt{S}(v, e)$, each with probability 1. For any set $V' \subseteq V$, let $W_{V'}$ denote the world where the tuples corresponding to V' are present. For any subset of nodes, V', we show that V' is an independent set if and only if $W_{V-V'}$ satisfies Q_1, i.e., all edges are present in its node-complement. Since $f(N) = V - N$ is one-to-one, the number of possible worlds that satisfy Q_1 are exactly the number of independent sets, thus completing the reduction. Now, if N is an independent set, then for any edge (u, v), it must be the case that at least one of u or v is in $V - N$, else the set would not be independent, since it would contain an induced edge. Thus, every edge is present and Q is satisfied. If N is not independent, then there must be some edge (u, v) such that $u, v \in N$, hence neither of u, v is in $V - N$. Since this edge is missing, Q_1 cannot be satisfied. This completes the reduction. The hardness of Q_2 is based on a reduction from counting the set covers of a fixed size and is in the appendix. □

There is some work in showing that the patterns in the previous theorem capture the boundary of hardness.

Theorem 4.4.16 (COUNT(DISTINCT) Dichotomy)**.** *Let* $Q[\alpha(y)\ \theta\ k]$ *be a* `HAVING` *such that* α *is* `COUNT(DISTINCT)`*, then either (1)* Q *is* `COUNT(DISTINCT)`*-safe and so has* P *data complexity or (2)* Q *is not* `COUNT(DISTINCT)`*-safe and has* ♯P*-hard data complexity.*

Proof. Part (1) of the result is Theorem 4.4.14. We sketch the proof of (2) in the simpler case when

only tuple independent probabilistic tables are used in Q and defer a full proof to Appendix B.2. Assume the theorem fails, let Q be the minimal counter example in terms of subgoals; this implies we may assume that Q is connected and the skeleton of Q is safe. Since there is no safe plan projecting on y and only independent projects are possible, the only condition that can fail is that some subgoal does not contain y. Thus, there are at least two subgoals $R(\boldsymbol{x})$ and $S(\boldsymbol{z}, y)$ such that $y \notin \boldsymbol{x} \cup \boldsymbol{z}$ and $\boldsymbol{x} \cap \boldsymbol{z} \neq \emptyset$. Given a graph (V, E), we then construct a BID instance J exactly as in the proof of Prop. 4.4.15. Only the R relation is required to have probabilistic tuples, all others can set their probabilities to 1. □

Extending to BID databases requires more work because our technique of adding extra tuples with probability 1 does not work: doing so naively may violate a possible worlds key constraint. The full proof appears in Appendix B.2. It is straightforward to decide if a plan is COUNT(DISTINCT)-safe: the safe plan algorithm of Dalvi and Suciu [48, 135] simply tries only disjoint projects and joins until it is able to project away y or it fails.

4.4.5 SUM-safe and AVG-safe queries

To find SUM- and AVG-safe queries, we need to further restrict the class of allowable plans. For example, there are queries involving SUM on a single table that are ♯P-hard, e.g., the query $Q[\texttt{SUM}(y) = k] :- R(y)$ is already ♯P-hard. There are, however, some queries that can be evaluated efficiently:

Definition 4.4.17. *A HAVING query $Q[\alpha(y)\ \theta\ k]$ for $\alpha \in \{\texttt{SUM}, \texttt{AVG}\}$ is α-safe, if there is a safe plan P for the skeleton of Q such that π^D_{-y} in P and no proper ancestor of π^D_{-y} is π^I_{-x} for any x.*

The idea of the positive algorithm is that if the plan contains π^D_{-y}, i.e., each value for y is present disjointly. Let $a_1, \ldots, a_n$ be the y values returned by running the standard query $q(y)$ (adding y to the head of $\mathbf{sk}(Q)$). Now consider the query Q' where $\mathbf{sk}(Q') = q[y \to a_i]$ (substitute y with a_i). On this query, the value of y is fixed, so we only need to compute the multiplicity of a_i figure out if Q' is true. To do this, we use the COUNT algebra of Section 4.4.3 whenever q is safe.

Theorem 4.4.18. *If $Q[\alpha(y)\ \theta\ k]$ for $\alpha \in \{\texttt{SUM}, \texttt{AVG}\}$ is α-safe, then Q's evaluation problem is in* P*-time.*

Sketch. Since Q is α-safe, then there is a plan P satisfying Definition 4.4.17. The consequence of this definition is that on any possible world W, we have that the conjunctive query $q(y)$ ($q = \mathbf{sk}(Q)$) returns a single tuple (i.e., a single binding for y). This implies that the values are *disjoint*. So for a fixed positive integer a returned by $q(y)$, the predicate $\texttt{SUM}(y)\ \theta\ k$ depends only on the *multiplicity* of a. Hence, we can write:

$$\mu[Q] = \sum_{a \in S} \mu[Q_a[\texttt{COUNT}(*)\ \theta\ \frac{k}{a}]$$

Here, Q_a denotes that $\mathbf{sk}(Q_a) = q[y \to a]$, i.e., y is substituted with a in the body of Q_a. Since Q is α-safe, we have that $q[y \to a]$ is safe, and so by Theorem 4.4.8, each term can be computed with the `COUNT` algebra. Hence, we can compute the entire sum in polynomial time and so $\mu[Q]$. For `AVG`, it is slightly simpler: Since we are taking the value of m copies of a, we have that the `AVG` is a if $m > 0$ (else the query is false). Thus, we simply need to compute the probability that the value a exists with multiplicity greater than 1 (which can be handled by the standard `EXISTS` algebra).

□

Example 4.4.19 Consider $Q[\texttt{SUM}(y) > 10] \coloneq \texttt{R}(\text{`a'}; y), \texttt{S}(y, u)$. This query is `SUM`-safe, with plan $\pi^D_{-y}(\texttt{R}(\text{`a'}; y) \bowtie \pi^I_{-u}\texttt{S}(y, u))$.

Complexity We show that if a `HAVING` query without self joins is not `SUM`-safe then, it has ♯P-data complexity. `AVG` follows by essentially the same construction.

Proposition 4.4.20. *Let $\alpha \in \{\texttt{SUM}, \texttt{AVG}\}$ and $\theta \in \{\leq, <, =, >, \geq\}$ then $Q[\alpha(y)\ \theta\ k] \coloneq R(y)$ has ♯P-data complexity.*

Proof. We only show `SUM`, deferring `AVG` to the appendix. Consider when θ is $=$. An instance of ♯`SUBSET-SUM` is a set of integers $x_1, \ldots, x_n$ and our goal is to count the number of subsets $S \subseteq 1, \ldots, n$ such that $\sum_{s \in S} x_i = B$. We create the representation with schema $R(X;;P)$ satisfying $\texttt{R} = \{(x_1; 0.5), \ldots, (x_n; 0.5)\}$, i.e., each tuple present with probability 0.5. Thus, $\mu(Q) * 2^n$ is number of such S. Showing hardness for other aggregate tests follows from Lemma 4.4.10. □

Theorem 4.4.21. *Let $\alpha \in \{\texttt{SUM}, \texttt{AVG}\}$ and let $Q[\alpha(y)\ \theta\ k]$ be a `HAVING` query, then either (1) Q is α-safe and hence has P-time data complexity, or (2) Q is not α-safe and Q has ♯P-data complexity.*

We prove this theorem in Appendix B.3.

4.5 Generating a Random World

In this section, we give an algorithm (Alg. 4.5.2.1) to solve the *random possible world generation problem*, which informally asks us to generate a possible world $\tilde{W}$ such that $q(\tilde{W}, \tau) = s$, i.e., such that the value of q on $\tilde{W}$ is s. The probability that we generate a fixed world $\tilde{W}$ is exactly the probability of $\tilde{W}$ *conditioned* on the value of q being equal to s. Our solution to this problem is a key a subroutine in our FPTRAS for `SUM` (in Section 4.6), but it is also an interesting problem in its own right. As pointed out by Cohen *et al.* [35], a random world satisfying some constraints is useful for many debugging and related tasks.

4.5.1 Problem Definition

Definition 4.5.1. *Let J be a BID instance, q be a conjunctive query, and τ be an annotation function. A* BID random world generator *(simply, a* random generator*) is a randomized algorithm $\mathcal{A}$ that generates a possible world $\tilde{W} \in \mathcal{W}_J$ such that for any $s \in S$ we have*[9]*:*

$$\mu_{\mathcal{A}}[\tilde{W} = W] = \mu(W \mid q(W, \tau) = s)$$

where $\mu_{\mathcal{A}}$ emphasizes that the probability is taken over the random choices of the algorithm $\mathcal{A}$. Further, we require that $\mathcal{A}$ run in time poly($|J|, |S|$).

This definition says that the probability a world is generated is *exactly* the conditional probability of that instance (conditioned on the value of the query q being s). In this section, we show that when **sk**(Q) is safe then we can solve create a random generator for any BID instance and any annotation function.

4.5.2 Possible World Generation Algorithm

To describe our algorithm, we need a notation to record the intermediate operations of the safe plan on the marginal vectors, i.e., a kind of *lineage* or *provenance* for the semiring computation.

[9]Formally, if $\mathcal{W} = \emptyset$, then we require that that a random generator return a special value, $\perp$. This value is like an exception and will not be returned during the course of normal execution.

Algorithm 4.5.2.1 A random world generator for J_ϕ

Decl: RWHELPER(ϕ : semiring parse tree,
s : a semiring value)
returns a random world denoted $\tilde{W} \subseteq J_\phi$.

---

if ϕ is a leaf, i.e., $\phi = (t, \boldsymbol{m}^t)$ for some tuple t **then**
(* If $s \neq 0$ then this implies the tuple must be present. *)
if $s \neq \tau(t)$ **then return** $\{t\}$
elif $s = 0$ **then return** $\emptyset$ **else return** $\perp$
(* Inductive case *)
Let ϕ have label (OP, m^r) and children ϕ_1 and ϕ_2
with marginal vectors m^{ϕ_1} and m^{ϕ_2}, respectively.
if OP $= \oplus$ **then**
Choose (s_1, s_2) s.t. $s_1 + s_2 = s$ with probability $m^{\phi_1}[s_1]m^{\phi_1}[s_2]\frac{1}{m^\phi[s]}$
if OP $= \otimes$ **then**
Choose (s_1, s_2) s.t. $s_1 \cdot s_2 = s$ with probability $m^{\phi_1}[s_1]m^{\phi_1}[s_2]\frac{1}{m^\phi[s]}$
if OP $= \bigsqcup$ **then**
Choose $(s_1, s_2) = (s, 0)$ with probability $m^{\phi_1}[s_1]\frac{1}{m^\phi[s]}$
or $(s_1, s_2) = (0, s)$ with probability $m^{\phi_1}[s_2]\frac{1}{m^\phi[s]}$
(* Union the results of the recursive calls *)
return RWHELPER$(\phi_1, s_1) \cup$ RWHELPER(ϕ_2, s_2)

Here, we view a safe plan as computing the marginal vectors *and* as computing a symbolic semiring expression (essentially, a parse tree of the extensional computation performed by the plan).

Definition 4.5.2. *A* semiring parse tree ϕ *is a binary tree where a leaf is labeled with a pair* $(t, \boldsymbol{m}^t)$ *where* t *is a tuple and* $\boldsymbol{m}$ *is a marginal vector on* S*; and an internal node is labeled with a pair* $(\text{OP}, \boldsymbol{m})$ *where* OP $\in \{\oplus, \otimes, \bigsqcup\}$ *and* $\boldsymbol{m}$ *is a marginal vector.*

Given a safe plan P and a BID instance J with annotation τ, the parse tree associated to P and J is denoted $\phi(P, J, \tau)$. We think of $\phi(P, J, \tau)$ as a record of the computation of P on J. More precisely, $\phi(P, J, \tau)$ is a parse tree for the semiring expression that we compute given P and J using the rules of Definition 4.4.5. The operations in a safe plan are n-ary: we can, however, transform these n-ary operations into a binary parse tree in an arbitrary way, since the operations are associative. An example parse tree is shown in Figure 4.6. We observe that any safe plan can be mapped to a parse tree.

Algorithm 4.5.2.2 A random world generator for J

Decl: RandomWorld(ϕ : semiring parse tree,
J : A BID instance, s a semiring element
returns a random world of J denoted $\tilde{W}$.

Let $\tilde{W} \leftarrow$ RWHelper(ϕ, s) and $T = J - J_\phi$
for each $t \in T$ **do**
  Let $K(t) = \{t' \mid t[K] = t'[K]\} = \{t_1, \ldots, t_m\}$ with $p_i = \mu[t_i]$.
  Let $\{t_{k+1}, \ldots, t_m\} = K(t) \cap J_\phi$
  if $K(t) \cap \tilde{W} = \emptyset$ **then**
    select t_i from $i = 1, k$ with $\frac{p_i}{1-\sum_{j=k+1,m} p_j}$ and $\tilde{W} \leftarrow \tilde{W} \cup \{t_i\}$
  $T \leftarrow T - K(t)$
return $\tilde{W}$

Example 4.5.3 Figure 4.6 illustrates how $\phi(P, J, \tau)$ is constructed for a simple example based on SUM. Figure 4.6(a) shows a relation $R(A; B)$ where A is a possible worlds key. Our goal is to generate a random world such that the query $Q[\texttt{SUM}(y) = 6] :- R(x; y)$ is true. The skeleton of Q is safe and so has a safe plan, $P = \pi^I_{-x}(\pi^D_{-y}(R))$. Figure 4.6(a) also shows the intermediate tuples that are computed by the plan, along with their associated marginal vectors. For example, the marginal vector associated to t_1 is $\boldsymbol{m}^{t_1}[0, 1] = (1 - p_1, p_1)$. Similarly, the marginal vector for intermediate tuples like t_6 is $\boldsymbol{m}^{t_6}[1] = p_2$. At the top of the plan is the empty tuple, t_8, and one entry in its associated marginal vector, i.e., $\boldsymbol{m}^{t_8}[6] = p_1 p_5 + p_2 p_4$.

The parse tree $\phi(P, J, \tau)$ corresponding to P on the instance $J = \{R\}$ is illustrated in Figure 4.6(b). The bottom-most level of internal nodes have $\text{OP} = \bigsqcup$, since they encode the action of the disjoint projection π^D_{-y}. In contrast, the root has $\text{OP} = \oplus$, since it records the computation of the independent project, π^I_{-x}. As we can see, the parse tree simply records the computation and the intermediate results.

Algorithm Overview Our algorithm has two phases: (1) We first build a random generator for the tuples in the parse tree ϕ (defined formally below); this is Alg. 4.5.2.1. (2) Using the tuples generated in step (1), we select those tuples not in the parse tree and complete the generator for J; this is Alg. 4.5.2.2. To make this precise, we need the following notation:

Definition 4.5.4. *Given a semiring parse tree ϕ, we define* **tup**(ϕ) *inductively: if ϕ is a leaf corre-*

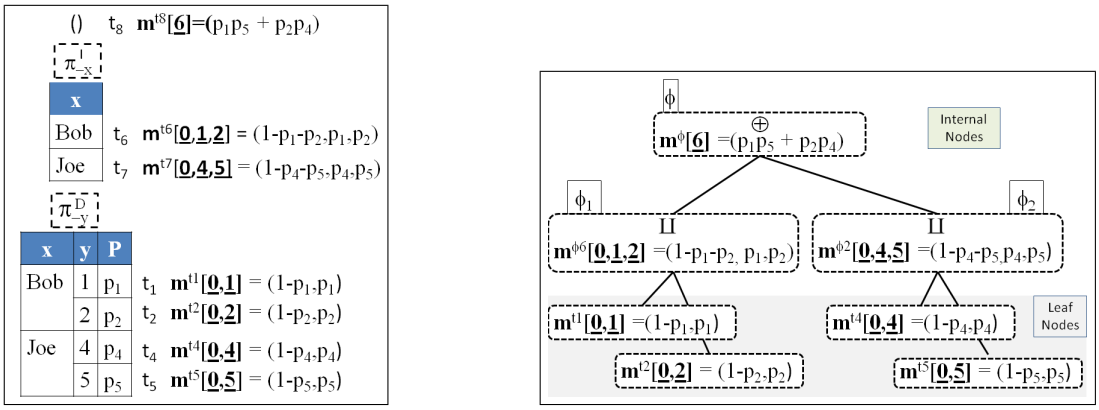


Figure 4.6: (a) A BID relation $R(A; B)$ used in Example 4.5.3 along with a safe plan $P = \pi^I_{-x}(\pi^D_{-y}(R))$. The extensional computation is in the semiring $\mathbb{N}$. (b) $\phi(P, \{R\})$ is shown where $P = \pi^I_{-x}(\pi^D_{-y}(R))$. The dotted boxes map to the nodes in the tree, described in Definition 4.5.2. In the figure, for readability, we only show the entry for 6 in the root. Also, the entries in the marginal vectors are real (rational) numbers and are written as expressions *only* for the sake of readability. There is a one-to-one correspondence between intermediate marginal vectors and nodes in the parse tree $\phi(P, J, \tau)$.

sponding to a tuple t, then $\mathbf{tup}(\phi) = \{t\}$. *Otherwise, ϕ has two child parse trees ϕ_1 and ϕ_2, then* $\mathbf{tup}(\phi) = \mathbf{tup}(\phi_1) \cup \mathbf{tup}(\phi_2)$. *We also consider* $\mathbf{tup}^+(\phi) = \mathbf{tup}(\phi) - \{t \mid \tau(t) = 0\}$, *i.e.,* $\mathbf{tup}^+(\phi)$ *is the set of tuples with non-zero annotations contained in ϕ.*

If P is a safe plan, then **tup** has a particular simple form:

Proposition 4.5.5. *Let P be a safe plan for q and J be a BID instance, then for any internal node ϕ_0 in $\phi(P, J, \tau)$ with children ϕ_1 and ϕ_2, we have that* $\mathbf{tup}(\phi_1) \cap \mathbf{tup}(\phi_2) = \emptyset$ *and* $\mathbf{tup}^+(\phi_1) \cap \mathbf{tup}^+(\phi_2) = \emptyset$.

Proof. We observe that an $\oplus$ or an $\coprod$ node is introduced only if there is a projection removing a variable x (Definition 4.4.4), in which case the tuples in $\mathbf{tup}(\phi_1)$ and $\mathbf{tup}(\phi_2)$ disagree on x, hence, are disjoint sets of tuples. Case two is that $\text{OP} = \otimes$, which is introduced only as a join of two tuples. In this case, $\mathbf{tup}(\phi_1)$ and $\mathbf{tup}(\phi_2)$ come from different relations (since there are no self joins in q). Thus, $\mathbf{tup}(\phi_1)$ and $\mathbf{tup}(\phi_2)$ have an empty intersection. The second statement follows since $\mathbf{tup}^+(\phi_i) \subseteq \mathbf{tup}(\phi_i)$ for $i = 1, 2$. □

For any parse tree ϕ, we can view the tuples in $\mathbf{tup}^+(\phi)$ as a BID instance that we denote J_ϕ (any subset of a BID instance is again, a BID instance). For a deterministic world W and a semiring expression ϕ, we write $\phi(W)$ to mean the semiring value of ϕ on world W, which is computed in the obvious way.

Step (1): A generator for J_ϕ We now define precisely the first step of our algorithm: Our goal is to construct a random world generator for the worlds induced by the BID instance J_ϕ. This is captured by the following lemma:

Lemma 4.5.6. *Let P be a safe plan for a query q, $\phi = \phi(P, J, \tau)$, and $J_\phi = \mathbf{tup}^+(\phi)$ then Alg. 4.5.2.1 is a random generator for J_ϕ for any annotation function τ.*

Proof. Let ϕ_0 be a subtree of $\phi(P, J, \tau)$. Then, given any $s \in S$, Alg. 4.5.2.1 is a random generator for J_{ϕ_0}. We induct on the structure of the parse tree ϕ. In the base case, ϕ_0 is a leaf node and our claim is straightforward: If $s = 0$, then we return the empty world. If $\tau(t) = s$, then we simply return a singleton world $\{t\}$ if $\tau(t) = s$. Otherwise, we have that $\tau(t) \neq s$, then the input is not well-formed and we return an exception ($\perp$) as required. This is a correct random generator, because our input is conditioned to be deterministic (i.e., μ has all the mass on a single instance).

We now write the probability that $\phi(W) = s$ in a way that shows that if we recursively can randomly generate worlds for subtrees, then we can make a random generator. Inductively, we consider an internal node ϕ with children ϕ_1 and ϕ_2. Assume for concreteness that $\text{OP} = \oplus$ (the argument for $\text{OP} = \otimes$ is identical and for $\text{OP} = \coprod$ is only a slight variation). Let W denote a world of J_ϕ. Then,

$$\mu[\phi(W) = s] \quad = \quad \mu[\phi_1(W) = s_1 \wedge \phi_2(W) = s_2 \mid s_1 + s_2 = s]$$

This equality follows from the computation of ϕ. We then simplify this expression using the fact that for $i = 1, 2$, ϕ_i's value is a function $\mathbf{tup}^+(\phi_i)$. Let $W_i = W \cap \mathbf{tup}^+(\phi_i)$, we get:

$$\mu[\phi_1(W_1) = s_1 \wedge \phi_2(W_2) = s_2 \mid s_1 + s_2 = s]$$

Observe that $\mu[s_1 + s_2 = s] = \mu[\phi(W) = s]$. Then, for any fixed s_1, s_2 such that $s_1 + s_2 = s$, we can then apply Bayes's rule and independence to get:

$$\frac{\mu[\phi_1(W_1) = s_1]\mu[\phi_2(W_2) = s_2]}{\mu[\phi(W) = s]}$$

Notice that W_1 (respectively, W_2) is a possible world of J_{ϕ_1} (respectively, J_{ϕ_2}) and so the inductive hypothesis applies. Now, by Prop. 4.5.5, the worlds returned by these worlds do not make conflicting choices. Since the recursive calls are correct, we just need to ensure that we pick (s_1, s_2) with the above probability. Examining Alg. 4.5.2.1, we see that we pick (s_1, s_2) with exactly this probability, since

$$\mu[\phi_1(W) = s_1 \wedge \phi_2(W) = s_2 \mid s_1 + s_2 = s] = \frac{m^{\phi_1}[s_1]m^{\phi_2}[s_2]}{m^{\phi}[s]}$$

This completes the proof. □

Example 4.5.7 We illustrate Alg. 4.5.2.1 using the data of Example 4.5.3. Our goal is to generate a random world such that the query $Q[\texttt{SUM}(y) = 6] :- R(x; y)$ is true. The algorithm proceeds top-down from the root ϕ. The entry for 6 is selected with probability equal to $p_1p_5 + p_2p_4$.

Assume we have selected 6, then we look at the child parse trees, ϕ_1 and ϕ_2: There are two ways to derive 6 with non-zero probability (1) the subtree ϕ_1 takes value 1 and ϕ_2 takes value 5, written $(\phi_1, \phi_2) = (1, 5)$ or (2) we set $(\phi_1, \phi_2) = (2, 4)$. We choose between these options randomly; we select $(\phi_1, \phi_2) = (1, 5)$ with probability equal to $\frac{p_1p_5}{p_1p_5+p_2p_4}$ (the conditional probability). Otherwise, we select $(\phi_1, \phi_2) = (2, 4)$. Suppose we have selected $(\phi_1, \phi_2) = (1, 5)$, we then recurse on the subtree ϕ_1 with value $s_1 = 1$ and the subtree ϕ_2 with value $s_2 = 5$.

Recursively, we can see that to set $\phi_1 = 1$, it must be that t_1 is present and t_2 is absent. Similarly, we conclude that t_4 must be absent and t_5 must be present. Hence, our random world is $\tilde{W} = \{t_1, t_5\}$. If we had instead chosen $(\phi_1, \phi_2) = (2, 4)$ then we would selected $\tilde{W} = \{t_2, t_4\}$. Notice that our algorithm never selects $(\phi_1, \phi_2) = (3, 3)$ (i.e., this occurs with probability 0). More generally, this algorithm *never* selects any invalid combination of tuple values.

Step (2): A generator for J We randomly include tuples in J that are not mentioned in ϕ, i.e., tuples in $J - J_\phi$. These are tuples that do not match any selection condition in the query, and can be freely added to $\tilde{W}$ without affecting the query result. Here, we need to exercise some care to not

insert two tuples with the same key into $\tilde{W}$, and so, we only consider tuples whose possible worlds key differs from those returned by Step (1). Formally, we prove the following lemma:

Lemma 4.5.8. *Let $\phi(P, J, \tau)$ be a parse tree for a safe plan P, a BID instance J, and an annotation τ. Then, given a random generator for J_ϕ, Alg. 4.5.2.2 is a random generator for J.*

Proof. We first use the random generator to produce a random world of J_ϕ, call it W_ϕ. Now, consider a tuple $t \in J - J_\phi$, let $K(t) = \{t' \mid t'[K] = t[K]\} = \{t_1, \ldots, t_m\}$, i.e., tuples distinct from t that share a key with t. If $K(t) \cap W_\phi \neq \emptyset$, then t cannot appear in this world because it is disjoint from the set of $K(t)$. Otherwise, $K(t) \cap W_\phi = \emptyset$, and let $K(t) - J_\phi = \{t_1, \ldots, t_k\}$ (without loss) with marginal probabilities $p_1, \ldots, p_k$, i.e., those key tuples not in $\mathbf{tup}^+(\phi)$. These tuples do not affect the value so all that matters is adding them with the correct probability, which is easily seen to be the conditional probability:

$$\mu[t_i \text{ is included }] = \frac{p_i}{1 - \sum_{j=k+1,\ldots,m} p_j}$$

This conditional simply says that it is conditioned on *none* of the tuples in $K(t) \cap J_\phi$ appearing. This is exactly Alg. 4.5.2.2 □

The main result We now state the main technical result of this section: It follows directly from the lemma above:

Theorem 4.5.9. *Let q be a safe conjunctive query, then Alg. 4.5.2.1 is a random generator for any BID instance J and annotation τ.*

An immediate consequences of Theorem 4.5.9 is that if the semiring S does not contain too many elements, then Alg. 4.5.2.1 solves the random possible world generation problem.

Corollary 4.5.10. *If q is safe and $|S| =$ poly($|J|$), then Alg. 4.5.2.1 solves the random possible world generation problem in time* poly($|J|$).

We use this corollary in the next section to design an FPTRAS for `SUM`.

4.6 Approximating HAVING queries with MIN, MAX and SUM

In this section, we study the problem of approximating HAVING queries. First, we describe an FPTRAS for having queries that have α = MIN where the test condition is $<$ or $\leq$, or α = MAX where the condition is one of $\{\geq, >\}$. This first FPTRAS applies to arbitrary such HAVING queries, including queries whose skeleton is unsafe. Second, we describe an FPTRAS for HAVING queries whose skeleton is safe, whose aggregate is SUM, and where the test condition is any of $<$, $\leq$, $>$, or $\geq$.

Our FPTRAS for SUM uses the random possible world generator of the previous section. These FPTRASes apply to a class of queries that we call (α, θ)-*apx-safe*. Additionally, we study the limits of any approach, and prove an approximation dichotomy for many (α, θ) pairs of HAVING queries without self joins: Either the above scheme is able to provide an FPTRAS and so the query is (α, θ)-apx-safe, or there is no FPTRAS: we call these queries (α, θ)-*hazardous*[10].

4.6.1 Background: Approximation of ♯P-Hard Problems

Although ♯P-problems are unlikely to be able to be solved exactly and efficiently, some problems have a strong approximation called a *Fully Polynomial Time Randomized Approximation Scheme* or FPTRAS [121], which is intuitively like a $1 + \varepsilon$ approximation.

Definition 4.6.1. *Given function* f *that takes an input* J *and returns a number* $\mathsf{f}(J) \in [0, 1]$*, where* J *is a BID instance, we say that an algorithm* $\mathcal{A}$ *is an* FPTRAS *for* f *if given any* $\delta > 0$*, a confidence, and any* $\varepsilon > 0$*, an error,* $\mathcal{A}$ *takes* J*,* ε*, and* δ *as input and produces a number denoted* $\tilde{\mathsf{f}}(J)$ *such that*

$$\mu_{\mathcal{A}}[\left|\mathsf{f}(J) - \tilde{\mathsf{f}}(J)\right| \leq \varepsilon\, \mathsf{f}(J)] > 1 - \delta$$

where $\mu_{\mathcal{A}}$ *is taken over the random choices of the algorithm,* $\mathcal{A}$*. Further,* $\mathcal{A}$ *runs in time polynomial in* ε^{-1}*,* $|W|$*, and* $\log \frac{1}{\delta}$*.*

This definition asks for a *relative approximation* [121], which means that if f is exponentially small, but non-zero, our algorithm is required to return a non-zero value. This is in contrast to an *absolute approximation*, that is allowed to return 0 (and could be constructed using naïve random

[10] Formally, we mean that the ♯BIS problem would have an FPTRAS, an unlikely outcome, [58, 59].

sampling). In this section, we fix a query Q and consider the function $f(J) = \mu_J(Q)$, where J is a BID instance. We study whether this function admits an FPTRAS.

We define three counting-like problems that are all ♯P-hard and will be of use later in this section:

Definition 4.6.2. *The ♯CLIQUE problem is given a graph (V, E), compute the fraction of the subsets of V that are cliques. The ♯BIS problem is given a bipartite graph (U, V, E), compute the fraction of of the subsets of $U \times V$ that are independent sets. The ♯KNAPSACK problem is given a set of positive integers $Y = \{y_1, \ldots, y_n\}$ and a positive integer value k, compute the fraction of sets $W \subseteq Y$ such that $\sum_{i \in W} y_i \leq k$.*

All three problems are ♯P-hard[11]. In a celebrated result, Jerrum and Sinclair [152] showed that ♯KNAPSACK *does* have an FPTRAS using a sophisticated Markov Chain Monte Carlo technique. It is believed that neither ♯CLIQUE nor ♯BIS have an FPTRAS. Interestingly, they are not equally hard to approximate (see Dyer *et al.* [58]). In particular, ♯BIS is a complete problem with respect to *approximation preserving reductions*. We do not need these reductions in their full generality, and simply observe that polynomial time computable 1-1 reductions (bijections) are approximation preserving. In this section, we say that a problem is ♯BIS-hard if there is a 1-1, polynomial-time reduction to ♯BIS.

The ♯KNAPSACK problem is related to the problem of computing HAVING queries with the aggregate functions SUM on a single table.

4.6.2 An FPTRAS for MIN with $\{\leq, <\}$ and MAX with $\{\geq, >\}$

Consider a query $Q[\texttt{MIN}(y) \leq k] \coloneq g_1, \ldots, g_l$ then an equivalent condition to $W \models Q$ is that $W \models q'$ where $q' \coloneq g_1, \ldots, g_l, y \leq k$. In other words, Q is equivalent to a conjunctive query, q', that contains an inequality predicate. As such, the standard algorithm for conjunctive queries on probabilistic databases [46, 78, 137] based on Karp-Luby [102] can be used. A symmetric argument can be used to find an FPTRAS for the aggregate test $(\texttt{MAX}, \geq)$. Thus, we get essentially for free the following theorem:

Theorem 4.6.3. *If $(\alpha, \theta) \in \{(\texttt{MIN}, \leq), (\texttt{MIN}, <), (\texttt{MAX}, \geq), (\texttt{MAX}, >)\}$ then $Q[\alpha(y)\ \theta\ k]$ has an FPTRAS.*

[11] We mean here that there is a $1-1$ correspondence with the counting variants of these problems, which are canonical ♯P-complete problems.

Although this theorem is easy to obtain, it is interesting to note that $Q[\texttt{MIN}(y) > k]$ has an FPTRAS *only* if $\mathbf{sk}(Q)$ is safe (as we show in Lemma 4.6.10). If $\mathbf{sk}(Q)$ is safe, then, we can compute its value exactly, so the FPTRAS is not very helpful. In contrast, Theorem 4.6.3 has no such restriction – $\mathbf{sk}(Q)$ can be an arbitrary conjunctive query. This is a striking example that approximation complexity may be more subtle than exact evaluation. In particular, an analog of Lemma 4.4.10 does not hold.

4.6.3 An FPTRAS for safe queries using SUM with $\{<, \leq, \geq, >\}$

The key idea of the FPTRAS is based on a generalization of Dyer's observation: for some $k \geq 0$, the query $Q[\texttt{SUM}(y) \leq k]$ is only hard to compute if k is very large. If k is small, i.e., polynomial in the instance size, then we can compute Q exactly. Dyer's idea is to *scale and round down the values, so that the y-values are small enough for exact computation*. The cost of rounding is that it introduces some spurious solutions, but not too many. In particular, the fraction of rounded solutions is large enough that if we can sample from the rounded solutions, then we can efficiently estimate the fraction of original solutions inside the rounded solutions.

To perform the sampling, we use Alg. 4.5.2.1 from the previous section (via Alg. 4.6.3.2). Pseudo-code for the entire FPTRAS is shown in Figure 4.6.3.1. We show only $(\texttt{SUM}, \leq)$ in detail, and explain informally how to extend to the other inequalities at the end of the section.

Theorem 4.6.4. *Let Q be a HAVING query $Q[\texttt{SUM}(y)\ \theta\ k]$ such that $\theta \in \{\leq, <\}$ and $\mathbf{sk}(Q)$ is safe, then Alg. 4.6.3.1 is an FPTRAS for Q.*

It is interesting to note that Theorem 4.6.4 implies that we can efficiently evaluate a *much larger* set of queries than the previous, complete exact algorithm (albeit only in an approximate sense). In particular, only a very restricted class of SUM-safe queries can be processed efficiently and exactly (c.f. Definition 4.4.17).

Our algorithm makes two technical assumptions: (1) in this section, unlike the rest of the paper, our semantics differ from SQL: In standard SQL, for a Boolean HAVING query q, if no tuples are returned by $\mathbf{sk}(Q)$ then $Q[\texttt{SUM}(y) \leq k]$ is false. In contrast, in this section, we assume that $Q[\texttt{SUM}(y) \leq 1]$ is true, even if $\mathbf{sk}(Q) = q$ is false, i.e., we choose the mathematical convention $\sum_{y \in \mathcal{Y}} = 0$, over SQL's choice, and (2) we make a *bounded odds* assumption[12] : for any tuple t there

[12]For example, that this rules our $p_t = 1$ for any tuple and allows any tuple to not be present with some probability.

Algorithm 4.6.3.1 An FPTRAS for SUM

Decl: SAMPLE(Q : a query $Q[\texttt{SUM}(y) \leq k]$ with a safe skeleton q,
an instance I, a confidence δ and error ε)
returns estimate of $\mu_I(Q)$.

Let body$(q) = \{g_1, \ldots, g_l\}$ and $n_i = |\mathbf{pred}(g_i)|$, i.e., the size of the ith relation.
Let $n = \prod_{i=1,\ldots,l} n_i$.
Let $Q^R[\texttt{SUM}(y) \leq n^2]$ with the same body as q (see below).
Let $\tau^R(y) = \lfloor \frac{n^2 y}{k} \rfloor$ and $y > k \mapsto n^2 + 1$
Construct an expression parse tree, $\phi = \phi(P, I, \tau^R)$ where P is a plan for q^R.
For $i = 1, \ldots m$ $W_i \leftarrow$ SAMPLEHELPER(ϕ, k)
(* Run m samples, for m a polynomial in δ, ε, n *)
return $\frac{|\{W_i | W_i \models Q^O\}|}{m} * \mu(Q^R)$ (* $\approx \frac{\mu(Q^O)}{\mu(Q^R)} \mu(Q^R) = \mu(Q^O)$ *)
(* Compute fraction of W_i that satisfy the original query Q^O. *)

Algorithm 4.6.3.2 Sampling Helper Routine

Decl: SAMPLEHELPER(ϕ: safe aggregate expression, b: a bound)
returns a world

Select $s \in 0, \ldots, b$ with probability $\frac{m^\phi[s]}{\sum_{s'} m^\phi[s']}$.
(* Select a final value for the query that is less than the bound b *)
return RANDOMWORLD(ϕ, s)

is exists $\beta > 1$ such that $\beta^{-1} \leq \frac{p_t}{1-p_t} \leq \beta$. These technical restrictions can be relaxed, but are chosen to simplify our analysis.

The Rounding Phase

The goal of the rounding phase is to produce a query and an annotation function that rounds the values in the instance down enough so that (1) the exact processing algorithms of Section 4.4.2 for SUM queries can be used, and (2) we can randomly generate a world using the algorithm of Section 4.5. Alg. 4.6.3.2 shows pseudo code for how these two steps are put together. The main result of this section is that the resulting algorithm is efficient (runs in time polynomial in the size of the BID instance J).

To get there, we construct two things: (1) an annotation function, τ^R, to do the rounding and (2) a query, Q^R, that uses the annotation function τ^R and the semiring $\mathbb{S}_{n^2+1}$ to compute the exact

distribution of the rounded sum in polynomial time.

The Annotation Function Let g be the first subgoal of q such that $\mathbf{var}(g) \ni y$ and $R = \mathbf{pred}(g)$, i.e., R is some relation containing y values. We scale down the values of y in R via the (rounding) annotation function denoted τ^R. Let $n = \prod_{g \in \mathbf{goal}(q)} |\mathbf{pred}(g)|$, i.e., the product of the sizes of all relations in q. Observe that n is polynomial in the instance size[13]. The rounded annotation function maps into the much smaller, rounded semiring $S^R = \mathbb{S}_{n^2+1}$. We define the rounded annotation function τ^R to agree everywhere with the original annotation function τ^O, *except* on g (the R relation): Here, $\tau^O_g(t) = t[y]$. In the rounded annotation function, we have $\tau^R_g(t) = \lfloor \frac{n^2}{k} t[y] \rfloor$, i.e., the y values are scaled down by a factor of n^2/k and rounded-down to the next highest integer. Additionally, if $t[y]$ is greater than k, then $\tau^R(t) = n^2 + 1$. Intuitively, this mapping is correct since if such a tuple is in the output of the query, then we are sure the summation is greater than k.

The Query We construct a rounded query $Q^R[\texttt{SUM}(y) \leq n^2]$ with the same body as Q^O. Let q be the skeleton of both Q^O and Q^R, i.e., $q = \mathbf{sk}(Q^R) = \mathbf{sk}(Q^O)$. We observe that since n^2 is polynomial in the instance size, the generic semiring algorithm of Section 4.4.2 can be used to compute the entire distribution $q(W, \tau^R)$ *exactly* in time polynomial in the size of the instance. Since we will always use Q^R with the rounded annotation function it makes sense to write $W \models Q^R$ if $q(W, \tau_R) \leq n^2$. Similarly, we will always use Q^O with the original annotation function so that it makes sense to write $W \models Q^O$ if $q(W, \tau_O) \leq k$.

Definition 4.6.5. *Let W be a possible world from some BID instance J. If $W \models Q^O$, then we call W an* original solution. *If $W \models Q^R$ then we call W a* rounded solution. *Further, denote the set of original solutions with W^O_J and rounded solutions with W^R_J:*

$$W^O_J = \{W \in \mathcal{W}_J \mid W \models Q^O\} \text{ and } W^R_J = \{W \in \mathcal{W}_J \mid W \models Q^R\}$$

We drop the subscript J when the BID instance is clear from the context.

We observe an essential property of our scheme: All original solutions are rounded solutions,

[13]Recall that the query is fixed so if the database contains m tuples then $n = m^{O(1)}$ where $O(1)$ hides a constant depending only on q, e.g., the number of subgoals suffices.

i.e., $W_J^O \subseteq W_J^R$. Formally,

Lemma 4.6.6. *For any possible world W, $W \models Q^O \implies W \models Q^R$, and more precisely, there exists a $\delta \in [0, n)$ such that $q(W, \tau^R) = q(W, \tau^O) - \delta$.*

Proof. Let $q = \mathbf{sk}(Q^O) = \mathbf{sk}(Q^R)$ and $\mathcal{V}$ be the set of all valuations for q. Let W be a possible world:

$$\begin{aligned} W \models Q^O \quad &\iff \quad \sum_{v \in \mathcal{V}: \mathbf{im}(v) \subseteq W} \tau^O(v(g)) \leq k \\ &\iff \quad \sum_{v \in \mathcal{V}: \mathbf{im}(v) \subseteq W} \frac{n^2}{k} \tau^O(v(g)) \leq n^2 \\ &\implies \quad \sum_{v \in \mathcal{V}: \mathbf{im}(v) \subseteq W} \tau^R(v(g)) + \delta_v \leq n^2 \\ &\iff \quad W \models Q^R \end{aligned}$$

Here, $\delta_v \in [0, 1)$ and accounts for the round-off of the floor function. Since $0 \leq \sum_v \delta_v < n$, we have the more precise statement. □

The importance of this lemma is that by sampling within the rounded solutions, we have a chance of hitting *any* original solution. Let $\tilde{W}$ be a random rounded solution created using Alg. 4.6.3.2, then let f be the Boolean-valued estimator (random variable) that takes value 1 iff $\tilde{W} \models Q^O$. It is not hard to see that this estimator satisfies:

$$\mathbf{E}_{\mathcal{A}}[\mathsf{f}] = \frac{\mu_J\left(W^O\right)}{\mu_J\left(W^R\right)}$$

Here, $\mathcal{A}$ is written to emphasize that the expectation is taken with respect to the (random) choices of Alg. 4.6.3.2. Importantly, this is exactly an individual trial of Alg. 4.6.3.1.

Analysis of the convergence

Using Alg. 4.6.3.2, we can efficiently conduct an individual (random) trial. The last technical piece to show that Alg. 4.6.3.1 is an FPTRAS, is to show that the number of trials m that are needed to guarantee that the estimator converges is small enough, i.e., $m = \text{poly}(|J|)$. The first lemma that we need is the standard $\{0, 1\}$-estimator lemma [121], which is an application of a Chernoff Bound.

Lemma 4.6.7 ([121])**.** *Let $m > 0$ be an integer. Given a sequence of independent Boolean-valued*

($\{0,1\}$) random variables $\mathsf{f}_1, \ldots, \mathsf{f}_m$ *with mean* $\mathbf{E}[\mathsf{f}]$*, then the estimator*

$$\mathsf{f}_m = \frac{1}{m} \sum_{i=1,m} \mathsf{f}_i$$

achieves a relative error of ε *with probability* $1 - \delta$ *for some* $m = O(\mathbf{E}[\mathsf{f}]^{-1}\varepsilon^{-2} \log \delta^{-1})$.

Observe that the estimator used in Alg. 4.6.3.1 is exactly of this type. The second lemma that we need is that the probability mass of the original solutions contained in the rounded solutions is "big enough" so that our sampling scheme will converge quickly.

Lemma 4.6.8. *Let* Q^R *and* Q^O *defined as above,* J *be a BID instance, and* μ_J *be* J*'s induced probability measure, then,*

$$(n+1)^{-1}\beta^{-1} \leq \frac{\mu_J(W^O)}{\mu_J(W^R)} \leq 1$$

where $n = \prod_{g \in \mathbf{goal}(q)} |\mathbf{pred}(g)|$.

This lemma is the technical heart of the argument: it intuitively places bounds on the variance of our estimate. We give a full proof in Appendix B.4. The importance of Lemma 4.6.8 is that it shows that $\mathbf{E}[\mathsf{f}] = \frac{\mu_J(W^O)}{\mu_J(W^R)} \geq n^{-1}\beta$, and so, applying Lemma 4.6.7, we see that we need at most $m = O(n\beta^{-1}\varepsilon^{-2} \log \delta^{-1})$ samples. We observe that a relative estimate for $\mathbf{E}[\mathsf{f}]$ implies that we have a relative estimate for $\mathbf{E}[\mathsf{f}]\mu_J(W^R) = \mu_J(W^O)$, the probability that we want to estimate. Thus, the algorithm is efficient as long as the the number of samples is bounded by a polynomial in $|J|$; a sufficient condition for this to hold is $\beta^{-1} = \text{poly}(|J|)$ which follows from the bounded odds assumption. Thus, under the bounded odds assumption with $\beta = \text{poly}(|J|)$, we have:

Theorem 4.6.9. *Let* Q *be a* `HAVING` *query* $Q[\alpha\ \theta\ k]$ *with* $\alpha =$ `SUM` *and* $\theta \in \{<, \leq, >, \geq\}$*, if the skeleton of* Q *is safe then* Q *has an* FPTRAS.

Extending to Other Inequalities A virtually identical argument shows that $\theta =$ ' $<$ ' has an FPTRAS. To see that $\geq$ has an FPTRAS with `SUM` on tuple independent database, the key observation is that we can compute a number $M = \max_W q(W, \tau)$. Then, we create a new BID instance $\bar{J}$ where each tuple $t \in J$, we map t to t' where $t = t'$ except that $t[P] = 1 - p$. We then ask the query $Q[\texttt{SUM}(y) < M - k]$, which is satisfied precisely on a world W when $Q[\texttt{SUM}(y) \geq k]$.

4.6.4 The Limits of Any Approach and a Dichotomy

We now study the limit of any approach to approximating HAVING queries. We see two interesting phenomenon: (1) the approximation depends not only the aggregate, but also the test. For example, $Q[\texttt{MIN} \leq k]$ has an FPTRAS while, in general, $Q[\texttt{MIN}(y) \geq k]$ does not. (2) The introduction of self joins results in problems that are believed to be harder to approximate than those without self joins; this suggests a more interesting complexity landscape for approximate query evaluation than exact query evaluation [59].

In this section, we only consider $\theta \in \{=, <, \leq, >, \geq\}$, i.e., we omit $\neq$ from consideration. To compactly specify aggregate tests, e.g., $(\texttt{MIN}, >)$, we write (α, Θ_0) where α is an aggregate and Θ_0 is a set of tests, i.e., $\Theta_0 \subseteq \{=, <, \leq, >, \geq\} = \Theta$; (α, Θ_0) is a short hand for the set $\bigcup_{\theta \in \Theta_0} \{(\alpha, \theta)\}$. We let $\Theta_{\leq} = \{\leq, <, =\}$ and $\Theta_{\geq} = \{\geq, >, =\}$. With this notation, we can state our first lemma.

Lemma 4.6.10. *Let (α, θ) be in $\{(\mathit{MIN}, \Theta_{\geq}), (\mathit{MAX}, \Theta_{\leq}), (\mathit{COUNT}, \Theta_{\leq}), (\mathit{SUM}, \Theta_{\leq})\}$ then the following HAVING query is $\sharp$BIS-hard:*

$$Q_{BIS}[\alpha(y)\ \theta\ k] :- R(x), S(x, y), T(y)$$

Let $Q[\alpha(y)\ \theta\ k]$ be a HAVING query such that **sk**(Q) *is not safe and consider only tuple-independent databases, then Q is $\sharp$BIS-hard.*

The second statement identifies the precise boundary of hardness for approximation over tuple independent databases.

Proof. We give a general construction that will be used in every reduction used to prove that $Q_{\texttt{BIS}}$ is $\sharp$BIS-hard. Given a bipartite graph (U, V, E), we create an instance of three relations R, S and T. The skeleton of our query in the reduction is $R(x), S(x, y), T(y)$. Without loss, we assume that U, V are labeled from $1, \ldots |U| + |V|$. Here, we encode a bipartite graph with $u \in U \mapsto R(u)$ and $v \in V \mapsto T(v)$, we assign each of these tuples probability 0.5. We let S encode E. It is not hard to see that there is a bijection between possible worlds and subsets of the graph. In particular, if a possible world corresponds to an independent set then no tuples are returned. We now add in a deterministic set of tuples, i.e., all probabilities are 1, as $\{R(a), S(a, a), T(a)\}$ for some a that we will

set below. These tuples are always present in the answer. Actually, *only* these tuples are present in the output if and only if this world encodes a bipartite independent set.

To see the reduction for MAX, set $a = 0$. We observe that $\texttt{MAX}(y) \leq 0$ if and only if the only tuple returned are the 0 tuples, i.e., a bipartite independent set. For MIN let $a = |U| + |V| + 1$, now check if $\texttt{MIN}(y) \geq a$. The $\texttt{COUNT}(y) \leq 1$ if only the a tuples are present. Similarly, SUM follows by setting $a = 1$ and ensuring all values are encoded higher. Thus, the bijection of the solution sets is the same.

Claim (2), that this reduction works for any unsafe query, follows by a result of Dalvi and Suciu [46] that shows that if a skeleton is not safe over tuple independent databases, it must *always* contain the $R(x), S(x, y), T(y)$ pattern used in this reduction. All other relations can contain a single tuple. This works because our reductions do not care about where the distinguished variable y falls, so we can set everything to 1 (or 0) in another relation. □

As a consequence of this lemma, the positive results of this paper, and the completeness of the safe plan algorithm of Dalvi and Suciu [46], we have the following:

Theorem 4.6.11. *Assume that* $\sharp\texttt{BIS}$ *does* not *have an* FPTRAS. *Let* (α, θ) *be one of* $(\texttt{MIN}, \Theta)$, $(\texttt{MAX}, \Theta)$, $(\texttt{COUNT}, \Theta_{\leq})$, *or* $(\texttt{SUM}, \Theta_{\leq})$ *then for any* HAVING *query* $Q[\alpha(y)\ \theta\ k]$ *over a tuple independent database J, either (1) the query evaluation problem can be approximated in randomized polynomial time and we call it* (α, θ)-apx-safe or *(2) the query evaluation problem does not have an* FPTRAS *and we call it* (α, θ)-hazardous. *Further, we can decide in which case Q falls in polynomial time.*

In some cases deciding in which case a query falls is trivial, e.g., a $(\texttt{MIN}, \leq)$ HAVING query is always (α, θ)-safe. In the cases that the decision is non-trivial, we can reuse the safe plan algorithm of Dalvi and Suciu [46] (applied to $\mathbf{sk}(Q)$). An immediate consequence of this dichotomy is a trichotomy which is obtained by combining the relevant theorem from Section 4.4 with the above theorem. For example, to get a trichotomy for the class of $(\texttt{COUNT}, \leq)$-HAVING queries, we combine Theorem 4.4.9 with the above theorem.

It is interesting to note that our positive algorithms work for arbitrary safe plans over BID databases. However, it is not immediately clear that the known hardness reductions (based on polynomial interpolation [48]), can be used to prove approximate hardness. Further, we leave the case of COUNT and SUM with $\{>, \geq\}$ open.

sk(Q)	($\texttt{MIN}, \Theta_<$), ($\texttt{MAX}, \Theta_>$)	($\texttt{MIN}, \Theta_>$), ($\texttt{MAX}, \Theta_<$), ($\texttt{COUNT}, \Theta$)
safe	safe (P, Theorem 4.4.8)	safe (P, Theorem 4.4.8)
not safe	apx-safe (FPTRAS, Theorem 4.6.3))	hazardous (no FPTRAS, Theorem 4.6.11)

Figure 4.7: Summary of results for MIN, MAX and COUNT. They form a trichotomy over tuple independent databases.

4.7 Summary of Results

Figure 4.7 summarizes our results for MIN, MAX and COUNT. If we restrict to HAVING queries over tuple independent instances the lines of the table are crisp and form a trichotomy: any such HAVING query cleanly falls into exactly one bucket. The positive results we have shown hold for all BID database. Over general BID databases, however, we have only established the weaker negative result that there exists *some* hard query when the skeleton is unsafe[14]. For example, the query $R(x;y), S(y)$ is known to be ♯P-hard [48]. Our results show that $Q[\texttt{COUNT}(y) \geq y] :\!- R(x;y), S(y)$ is ♯P-hard, but leave open whether it has an FPTRAS.

The state of the results with (SUM, <) over tuple-independent databases is more interesting: If Q is SUM-safe, then its evaluation is in P-time (Theorem 4.4.18). If Q is not SUM-safe, but **sk**(Q) is safe then Q is ♯P-hard (Theorem 4.4.21), but does admit an FPTRAS (Theorem 4.6.9). We call Q (SUM, <)-apx-safe. If **sk**(Q) is not safe, then evaluating Q is ♯BIS-hard (Theorem 4.6.11), and so likely has no FPTRAS. We call Q (SUM, <)-hazardous. We now show that with the addition of self joins, even a simple pattern becomes as hard to *approximate* as ♯CLIQUE, which is as hard to approximate as any problem in ♯P. This is interesting because it points out that the complexity of approximation may be richer than we have explored in this paper:

Lemma 4.7.1. *Let* (α, θ) *be in* $\{(\texttt{MIN}, \Theta_\leq), (\texttt{MAX}, \Theta_\geq), (\texttt{COUNT}, \Theta_\leq), (\texttt{SUM}, \leq)\}$ *and consider the* HAVING *query:*

$$Q_{\texttt{CLIQUE}}[\alpha(y)\ \theta\ k] :\!- R(x), S(x,y), R(x)$$

then $Q_{\texttt{CLIQUE}}$ *is as hard to approximate as* ♯CLIQUE*.*

Proof. The input instance of ♯CLIQUE is $G = (V, E)$: For each $v \in V$, we create a tuple $R(v)$ that has

[14]It is an open problem to extend these results to all BID databases.

probability $\frac{1}{2}$ and E encodes exactly the *complement* (symmetrically closed) edge relation; Here, $(v, v) \notin E$. Notice that a possible world is simply a subset of R. If in a possible world, $q = \mathbf{sk}(Q)$ is satisfied then, this implies there is some pair of nodes (u, v) that are not connected by an edge in G and so W does not represent a clique. Hence the query is false precisely when $\sharp$CLIQUE is true. Using exactly the same encoding as we used in the previous proof, we can then test the probability of this condition. □

Chapter 5

VIEW-BASED TECHNIQUE I: MATERIALIZED VIEWS

The technique of materialized views is widely used to speed up query evaluation in relational optimizers. Early relational optimizers were restricted to using simple indexes (which are materalized views that contain a simple projection) [161], while modern optimizers can use materialized views that are defined by aribitrary SQL [5]. In general, materialized views can provide dramatic improvements in query performance for expensive queries. Materialized views are a form of caching: instead of computing the query from scratch, we precompute a portion of the information needed by a query, and use this information to reduce our work at query time. In probabilistic databases, query evaluation is not only practically expensive, but theoretically expensive (♯P-hard). As a result, it is natural to suspect (and it is indeed the case) that materialized views have an even greater impact on optimizing probabilistic queries than they do for standard, relational database queries. In this chapter, we discuss how to apply materialized view techniques to probabilistic databases.

The major, new challenge in using materialized views in a probabilistic database query optimizer is that views may contain complex correlations. One way to track this correlation is using a complete approach based on *lineage* [51, 137, 148] which effectively tracks *every* derivation for a tuple in the output. Using the complete approach allows any query to be used with any view, but it does not allow the large performance gains that we expect from materialized views. The reason is that the bottleneck in query answering is *not* computing the lineage, but is in performing the probabilistic inference necessary to compute the probability of an answer tuple. In this chapter, we study a more aggressive approach that avoids recomputing the lineage at query time, and so provides greater gains than the complete approach.

The key to our technique is a novel static analysis of the view and query definitions that allows us to determine how the tuples in the view are correlated. An important desideratum for our check is that it is only a function of the view and query definitions, and not the data, which means it can be used in a query optimizer. The property that we are testing is over an (uncountably) infinite set

of database, i.e., do the claimed independences hold for *any* probabilistic relational database? As a result, it is not clear that this test is even decidable. Our main technical result is that this test is decidable for conjunctive views, and is in Π_2P, the second level of the polynomial hierarchy [128, pg. 433]. In fact, the decision problem is complete for Π_2P. To cope with this high complexity, we provide efficient approximations, i.e., polynomial-time tests that are sound, but not complete. We show that for a restricted class of query and view definitions these simple, efficient tests actually are complete. One important special case is when a view *V* is *representable*, which operationally means that it can be treated by the query optimizer as as if it were *BID* table, and so, we can use all the query optimization techniques in the MystiQ system, notably, safe plans [46].

We validate our solutions using data given to us by iLike.com [40], a service provided by the music site GarageBand [39], which provides users with recommendations for music and friends. iLike has three characteristics which make it a good candidate for materialized views. First, the data are uncertain because the recommendations are based on similarity functions. Second, the database is large (tens of gigabytes) and backs an interactive website with many users; hence, query performance is important. Third, the database hosts more than one service, implying that there are integration issues. Probabilistic materialized views are a tool that addresses all three of these issues. Interestingly, materialized views are present in iLike. Since there is no support for uncertain views, the materialized views are constructed in an *ad hoc* manner and require care to use correctly. Our experiments show that 80% of the materialized views in iLike are representable by BID tables and more than 98.5% of their workload could benefit from our optimizations. We also experiment with synthetic workloads and find that the majority of queries can benefit from our techniques. Using the techniques described in this paper, we are able to achieve speed ups of more than three orders of magnitude on large probabilistic data sets (e.g., TPC 1G with additional probabilities).

5.1 Motivating Scenario and Problem Definition

Our running example is a scenario in which a user, Alice, maintains a restaurant database that is extracted from web data and she wishes to send data to Bob. Alice's data are uncertain because they are the result of *information extraction* [84, 96, 126] and *sentiment analysis* [62]. A natural way for her to send data to Bob is to write a view, materialize its result and send the result to Bob.

Chef	Restaurant	**P**	
TD	D. Lounge	0.9	(w_1)
TD	P. Kitchen	0.7	(w_2)
MS	C. Bistro	0.8	(w_3)

W(Chef,Restaurant;;**P**) (WorksAt)

Restaurant	Dish
D. Lounge	Crab Cakes
P. Kitchen	Crab Cakes
P. Kitchen	Lamb
C. Bistro	Fish

S(Restaurant,Dish) (Serves)

Chef	Dish	Rating	**P**	
TD	Crab Cakes	High	0.8	(r_{11})
		Med	0.1	(r_{12})
		Low	0.1	(r_{13})
TD	Lamb	High	0.3	(r_{21})
		Low	0.7	(r_{22})
MS	Fish	High	0.6	(r_{31})
		Low	0.3	(r_{32})

R(Chef,Dish;Rating;**P**) (Rated)

Figure 5.1: Sample Restaurant Data in Alice's Database. In `WorksAt` each tuple is *independent*. There is no uncertainty about `Serves`. In `Rated`, each (Chef,Dish) pair has one true rating. All of these are BID tables.

Sample data is shown in Figure 5.1 for Alice's schema that contains three relations described in BID syntax: `W` (`WorksAt`), `S` (`Serves`) and `R` (`Rating`). The relation `W` records chefs, who may work at multiple restaurants in multiple cities. The tuples of `W` are extracted from text and so are uncertain. For example, ('TD', 'D. Lounge') (w_1) in `W` signifies that we extracted that 'TD' works at 'D.Lounge' with probability 0.9. Our syntax tells us that all tuples are independent because they all have different possible worlds keys. The relation `R` records the rating of a chef's dish (e.g. 'High' or 'Low'). Each (Chef,Dish) pair has only one true rating. Thus, r_{11} and r_{13} are *disjoint* because they rate the pair ('TD', 'Crab Cakes') as both 'High' and 'Low'. Because distinct (Chef,Dish) pair ratings are extracted independently, they are associated to ratings independently.

Semantics In `W`, there are 2^3 *possible worlds*. For example, the probability of the singleton subset {('TD', 'D. Lounge')} is $0.9 * (1 - 0.7) * (1 - 0.8) = 0.054$. This representation is called a p-?-table [82], ?-table [51] or tuple independent [46]. The BID table `R` yields $3 * 2 * 3 = 18$ possible worlds since each (Chef,Dish) pair is associated with at most one rating. When the probabilities shown sum to 1, there is at least one rating for each pair. For example, the probability of the world $\{r_{11}, r_{21}, r_{31}\}$ is $0.8 * 0.3 * 0.6 = 0.144$.

Representable Views

Alice wants to ship her data to Bob, who wants a view with all chefs and restaurant pairs that make a highly rated dish. Alice obliges by computing and sending the following view:

$$V_1(c, r) \text{ :- } \mathtt{W}(c, r), \mathtt{S}(r, d), \mathtt{R}(c, d; \text{'High'}) \tag{5.1}$$

In the following example data, we calculate the probability of a tuple appearing in the output both numerically in the **P** column and symbolically in terms of other tuple probabilities of Figure 5.1. We calculate the probabilities symbolically only for exposition; the output of a query or view is a set of tuples matching the head with associated probability scores.

Example 5.1.1 [Output of V_1 from Eq. (5.1)]

	C	R	**P**	(Symbolic Probability)
t_1^o	TD	D. Lounge	0.72	$w_1 r_{11}$
t_2^o	TD	P.Kitchen	0.602	$w_2(1-(1-r_{11})(1-r_{21}))$
t_3^o	MS	C.Bistro	0.32	$w_3 r_{31}$

The output tuples, t_1^o and t_2^o, are *not* independent because both depend on r_{11}. This lack of independence is problematic for Bob because a BID instance cannot represent this type of correlation; Hence, we say V_1 is not a *representable view*. For Bob to understand the data, Alice must ship the lineage of each tuple. For example, it would be sufficient to ship the symbolic probability polynomials in Example 5.1.1.

Consider a second view where we can be much smarter about the amount of information necessary to understand the view. In V_2, Bob wants to know which working chefs make and serve a highly rated dish:

$$V_2(c) \text{ :- } \mathtt{W}(c, r), \mathtt{S}(r, d), \mathtt{R}(c, d; \text{'High'}) \tag{5.2}$$

Example 5.1.2 [Output of V_2 from Eq. (5.2)]

c	**P**	(Symbolic Probability)
TD	0.818	$r_{11}(1-(1-w_1)(1-w_2)) + (1-r_{11})(w_2 r_{21})$
MS	0.48	$w_3 r_{31}$

Importantly, Bob can understand the data in Example 5.1.2 with no auxiliary information because the set of events contributing to 'TD' and those contributing to 'MS' are *independent*. It can be shown that over any instance, all tuples contributing to distinct c values are independent. Thus, V_1 is a *representable view*. This motivates us to understand the following fundamental question:

Problem 1 (View Representabilty)**.** *Given a view V, can the output of V be represented as a BID table?*

It is interesting to note that efficiency and representability are distinct concepts: Computing the output of V_1 can be done in polynomial time, but its result is not representable. On the other hand, computing V_2 can be shown to be $\sharp$P-Hard [46, 135], but V_2 is *representable*. To process V_2, we must use Monte Carlo procedures (e.g., [137]), which are orders of magnitude more expensive than traditional SQL processing. We do not discuss evaluation further because our focus is not on efficiently evaluating views, but on representing the output of views.

Partially Representable Views

To optimize and share a larger class of views, we want to use the the output of views that are not *representable*. The output of a non-representable view still has meaning: It contains the marginal probabilities that each tuple appears in the view but may not describe how all the tuples in the view correlate. Consider the output of V_1 in Example 5.1.1, although this view is not a BID table, there is still a lot of correlation information in the view definition: It is not hard to see that tuples that agree on c are correlated, but tuples with different c values are independent.

To build intuition, notice that when a view V is a BID table, then, denoting $K = Key(V)$, the following two properties hold: for any set of tuples, if any two tuples in the set differ on at least one of the K attributes, then the set of tuples is independent, and any two tuples that agree on the K attributes are disjoint. If the view V is not a BID table, but an arbitrary probabilistic relation, then we may still be able to find two sets of attributes, L and K, that satisfy these two properties separately. Formally:

Definition 5.1.3. *Let (W, μ) be a probabilistic database over a single relation V.*

- *We say that (W, μ) is L-block independent, where $L \subseteq Attr(V)$, if any set of tuples $\{t_1, \ldots, t_n\}$ s.t. $t_i.L \neq t_j.L$, $1 \leq i < j \leq n$, is independent.*

- *We say that* (W, μ) *is K-block disjoint, where* $K \subseteq Attr(V)$*, if any two tuples* t, t' *s.t.* $t.K = t'.K$ *are disjoint. Equivalently, K is a key in each possible world of V.*

Example 5.1.4 Recall the view V_1 in Eq. (5.1), whose natural schema is $V(C, R)$. It is has a partial representation (L, K) with $L = \{C\}$, $K = \{C, R\}$. Tuples that differ on C are independent, but those that agree on C but differ on R may be correlated in unknown ways. Thus, the materialized view does have some meaning for Bob, but does not contain sufficient information to completely determine a probability distribution on its possible worlds.

Trivially, any view $V(H)$ can be partially represented by letting $L = \emptyset$ and $K = H$. At the other extreme, a BID table is one in which $L = K$. Intuitively, we want a "large" L and a "small" K. Thus the interesting question is: What is the complexity to decide, for a given K, L, if a view V is *partially representable?* This is a generalization of Problem 3, and this is one of our main technical results in this chapter.

Using Views to Answer Queries

In an information exchange setting, we do not have access to the base data and so for partially representable views may not know how all tuples are correlated. Thus, to answer a query Q, we need to ensure that the value of Q does not depend on correlations that are not captured by the partial representation. To illustrate, we attempt to use the output of V_1, a partially representable view, to answer two queries.

Example 5.1.5 Suppose Bob has a local relation, $\mathtt{L}(d; r)$, where d is a possible worlds key for $\mathtt{L}$. Bob wants to answer the following queries:

$$Q_u(d) \text{ :- } \mathtt{L}(d; r), V_1(c, r) \text{ and } Q_n(c) \text{ :- } V_1(c, r)$$

Since the materialized view V_1 does not uniquely determine a probability distribution on its possible worlds, it is not immediately clear that Bob can answer these queries without access to Alice's database and to V_1's definition. However, the query Q_u is uniquely defined. For any fixed d_0, the set

of r_i values such that $\mathrm{L}(d_0; r_i)$ partition the possible worlds:

$$\mu(Q_u(d_0)) = \sum_{r_i : I \models \mathrm{L}(d_0; r_i)} \mu(\mathrm{L}(d_0; r_i) \wedge \exists c\ V_1(c, r_i))$$

The partial representation $V_1(C; R; \emptyset)$, tells us that distinct values for c in V_1 are independent. Thus, each term in the summation is uniquely defined implying Q_u is uniquely defined. In contrast, Q_n is not uniquely defined because Q_n('TD') is true when either of t_1^o or t_2^o are present; our partial representation does not capture the correlation of the pair (t_1^o, t_2^o).

This example motivates the following problem:

Problem 2 (View Answering)**.** *Let* (L, K) *be a partial representation for a view* V*. Given a query* Q *using* V*, is the value of* Q *uniquely defined?*

We will solve this problem in the case when V is a conjunctive view and Q is any monotone Boolean property. A special case of this general result is when Q is specified by a conjunctive query.

Relationship with Query Evaluation As we noted earlier, efficient query evaluation for a view and its degree of representability are distinct concepts. When a query has a PTIME algorithm, it is called a *safe query* [46]. It can be seen that the super-safe plan optimization of Chapter 3.3 are those plans which are representable, safe, and do not contain self joins. Thus, progress on the view answering problem allows us to extend this technique to a larger class of subplans. For the case of SQL-like queries (conjunctive queries), the results in this chapter give a complete solution for the view answering problem which allows us to build a query optimizer for large-scale probabilistic databases.

5.2 Query Answering using Probabilistic Views

We begin by discussing a very general problem: how to find a partial representation (L, K) for an arbitrary probabilistic relation; we show that there is always a unique maximal choice for L in the partial representation such that a weaker independence property holds (2-independence). Then, we show that if a view V is defined in first-order logic (FO), its output has a unique maximal L such that

our stronger property of total independence holds. We then show that if V is defined by a conjunctive query, then we can find L with a $\Pi_2\mathsf{P}$ algorithm (and moreover that the related decision problem is $\Pi_2\mathsf{P}$-complete). Finally, we discuss the problem of answering queries using a partial represented views. The technical issue is to decide if a query using a view has a uniquely defined value[1]. We show that this property is decidable for conjunctive queries and conjunctive views, and moreover is complete for $\Pi_2\mathsf{P}$ as well.

5.2.1 Partial Representation of a Probabilistic Table

In this section we will not assume that a table V is a block disjoint-independent (BID) table, but rather allow it to be an arbitrary probability space over the set of possible instances for V. The intuition is that V is a view computed by evaluating a query over some BID database: the output of the view is, in general, not a BID table, but some arbitrary probabilistic database $(\mathcal{W}, \mu)$.

In general, a probabilistic table V may admit more than one partial representation. For example, every probabilistic table V admits the trivial partial representation $L = \emptyset$ and $K = Attr(V)$, but this representation is not useful to answer queries, except the most trivial queries that check for the presence of a single ground tuple. Intuitively, we want a "large" L and a "small" K. It is easy to check that we can always relax the representation in the other way: if (L, K) is a partial representation, and L', K' are such that $L' \subseteq L$ and $K \subseteq K'$, then (L', K') is also a partial representation. Of course, we want to go in the opposite direction: increase L, decrease K. We will give several results in the following sections showing that a unique maximal L exists, and will explain how to compute it. On the other hand, no minimal K exists in general; we will show that the space of possible choices for K can be described using standard functional dependency theory.

It is not obvious at all that a maximal L exists, and in fact it fails in the most general case, as shown by the following example.

Example 5.2.1 Consider a probabilistic table $V(A, B, C)$ with three possible tuples:

---

[1]This presentation in this section is from *"Probabilistic query answering and query answering using Views"* in the Journal of Computer and System Sciences [49].

T :

A	B	C	
a	b	c	t_1
a	b'	c	t_2
a	b'	c'	t_3

and four possible worlds: $I_1 = \emptyset$, $I_1 = \{t_1, t_2\}$, $I_2 = \{t_2, t_3\}$, $I_3 = \{t_1, t_3\}$, each with probability $1/4$. Any two tuples are independent: indeed $\mu(t_1) = \mu(t_2) = \mu(t_3) = 1/2$ and $\mu(t_1t_2) = \mu(t_1t_3) = \mu(t_2t_3) = 1/4$. V is AB-block independent: this is because the only sets of tuples that differ on AB are $\{t_1, t_2\}$ and $\{t_1, t_3\}$, and they are independent. Similarly, V is also AC-block independent. But V is not ABC-block independent, because any two tuples in set $\{t_1, t_2, t_3\}$ differ on ABC, yet the entire set is not independent: $\mu(t_1t_2t_3) = 0$. This shows that there is no largest set L: both AB and AC are maximal.

A weaker result holds. For a set $L \subseteq Attr(V)$ we say that V is L-block 2-independent if any two tuples t_1, t_2 s.t. $t_1.L \neq t_2.L$ are independent.

Lemma 5.2.2. *Let V be a probabilistic table. Then there exists a maximal set L s.t. V is L-block 2-independent.*

Proof. We prove the following: if $L_1, L_2 \subseteq Attr(V)$ are such that V is L_i-block 2-independent for each $i = 1, 2$, then V is also L-block 2-independent, where $L = L_1 \cup L_2$. Indeed, let t_1, t_2 be two tuples s.t. $t_1.L \neq t_2.L$. Then either $t_1.L_1 \neq t_2.L_1$ or $t_1.L_2 \neq t_2.L_2$, hence t_1, t_2 are independent tuples. The largest set L claimed by the lemma is then the union of all sets L' s.t. V is L'-block 2-independent. □

Continuing Example 5.2.1 we note that V is ABC-block 2-independent, since any two of the tuples t_1, t_2, t_3 are independent.

5.2.2 Partial Representation of a probabilistic lineage database

Recall that if V is a view defined by an FO query over a BID database, then V can be expressed as probabilistic lineage database. In this section we study the partial representation of a probabilistic lineage database. Recall that a lineage database (λ, μ) consists of two parts: λ is a function that

assigns each tuple a Boolean expression and μ is a product probability space on the set of variables $\bar{X}$ that are used in the Boolean expressions of λ. Our main result in this section is the following: given the lineage function λ and a particular relation V, there exists a maximal set of attributes $L \subseteq \text{Attr}(V)$ such that for any product probability space μ, V is L-block independent. Thus, if V is a view defined by an FO query over a BID database, then this result shows us how to compute a good partial representation for the view.

Let $\bar{X} = \{X_1, \ldots, X_m\}$ be the variables used in the Boolean expressions in λ, and let $Dom(X_j)$ be the finite domain of values for the variable X_j, $j = 1, m$. The annotated table V consists of n tuples $t_1, \ldots, t_n$, each annotated with a Boolean expression $\lambda(t_1), \ldots, \lambda(t_n)$ obtained from atomic formulas of the form $X_j = v$, where $v \in Dom(X_j)$, and the connectives $\wedge$, $\vee$, and $\neg$. A *valuation* θ is a function $\theta : \bar{X} \rightarrow \bigcup_i Dom(X_j)$ s.t., $\theta(X_j) \in Dom(X_j)$, for $j = 1, m$.

The following definition is adapted from Miklau and Suciu [119].

Definition 5.2.3. *Let φ be a Boolean expression over variables $\bar{X}$. A variable X_j is called a* critical variable *for φ if there exists a valuation θ for the variables $\bar{X} - \{X_j\}$ and two values $v', v'' \in Dom(X_j)$ s.t. $\varphi[\theta \cup \{(X_j, v')\}] \neq \varphi[\theta \cup \{(X_j, v'')\}]$.*

For a simple illustration, suppose $Dom(X_1) = Dom(X_2) = Dom(X_3) = \{0, 1, 2\}$, and consider the Boolean expression

$$\varphi \quad \equiv \quad (X_1 = 0) \vee (X_1 = 1) \vee ((X_3 = 1) \wedge \neg(X_1 = 2))$$

Then X_2 is not a critical variable for φ, because it is not mentioned in the expression of φ. X_3 is also not a critical variable, because φ simplifies to $(X_1 = 0) \vee (X_1 = 1)$ (since $Dom(X_1) = \{0, 1, 2\}$). On the other hand, X_1 is a critical variable: by changing X_1 from 0 to 2 we change φ from *true* to *false*. In notation, if θ is the valuation $\{(X_2, 0), (X_3, 0)\}$, then $\varphi[\theta \cup \{(X_1, 0)\}] = true$, $\varphi[\theta \cup \{(X_1, 2)\}] = false$.

The main result in this section is based on the following technical lemma:

Lemma 5.2.4. *Let φ, ψ be two Boolean expressions. Then the following two statements are equivalent:*

- *For every product probability space μ on the set of variables $\bar{X}$, φ and ψ are independent events.*

- *φ and ψ have no common critical variables.*

Proof. The "if" direction is straightforward: if φ, ψ use disjoint sets of variables, then $\mu(\varphi \wedge \psi) = \mu(\varphi)\mu(\psi)$, hence they are independent for any choice of μ.

The "only if" direction was shown in [119] for the case when all variables X_j are Boolean, i.e. $|Dom(X_j)| = 2$. We briefly review the proof here, then show how to extend it to non-Boolean variables. Given a probability spaces $(Dom(X_j), \mu_j)$, denote $x_j = \mu_j(X_j = 1)$, hence $\mu_j(X_j = 0) = 1 - x_j$. Then $\mu(\varphi)$ is a polynomial in the variables $x_1, \ldots, x_m$ where each variable has degree ≤ 1. (For example, if $\varphi = \neg(X_1 \otimes X_2 \otimes X_3)$ (exclusive or) then $\mu(\varphi) = x_1x_2(1 - x_3) + x_1(1 - x_2)x_3 + (1 - x_1)x_2x_3 + (1 - x_1)(1 - x_2)(1 - x_3)$, which is a polynomial of degree 1 in x_1, x_2, x_3.) One can check that if X_j is a critical variable for φ then the degree of x_j in the polynomial $\mu(\varphi)$ is 1; on the other hand, if X_j is not a critical variable, then the degree of x_j in the polynomial $\mu(\varphi)$ is 0 (in other words the polynomial does not depend on x_j). The identity $\mu(\varphi)\mu(\psi) = \mu(\varphi \wedge \psi)$ must hold for any values of $x_1, \ldots, x_m$, because φ, ψ are independent for any μ. If X_j were a common critical variable for both φ and ψ then the degree of x_j in the left hand side polynomial is 2, while the right hand side has degree at most 1, which is a contradiction.

We now extend this proof to non-Boolean domains. In this case a variable X_j may take values $0, 1, \ldots, d_j$, for $d_j \geq 1$. Define the variables x_{ij} to be $x_{ij} = \mu(X_j = i)$, for $i = 1, \ldots, d_j$, thus $\mu(X_j = 0) = 1 - x_{1j} - x_{2j} - \cdots - x_{d_j j}$. As before $\mu(\varphi)$ is a polynomial of degree 1 in the variables x_{ij} with the additional property that if $i_1 \neq i_2$ then $x_{i_1 j}$ and $x_{i_2 j}$ cannot appear in the same monomial. We still have the identity $\mu(\varphi\psi) = \mu(\varphi)\mu(\psi)$, for all values of the variables x_{ij} (since the identity holds on the set $x_{ij} \geq 0$ for all i, j, and $\sum_i x_{ij} \leq 1$, for all j, and this set has a non-empty interior). If X_j is a critical variable for φ then $\mu(\varphi)$ must have a monomial containing some $x_{i_1 j}$; if it is also critical for ψ, then $\mu(\psi)$ has a monomial containing $x_{i_2 j}$. Hence their product contains $x_{i_1 j} \cdot x_{i_2 j}$, contradiction. □

We will use the lemma to prove the main result in this section. Let λ_V denote the lineage function λ restricted to tuples in V. Thus, (λ_V, μ) defines a probabilistic database with a single relation, V.

Theorem 5.2.5. *Let V be a table in a lineage database (λ, μ). Then there exists a unique maximal set of attributes L such that, for any product probability space μ, the lineage database (λ_V, μ) is L-block independent.*

Proof. Denote $t_1, \ldots, t_n$ the tuples V, and let $\lambda(t_1), \ldots, \lambda(t_n)$ their Boolean expressions annotations. Let L be a set of attributes. We prove the following:

Lemma 5.2.6. *The following three statements are equivalent:*

1. *For any μ, the pdb (λ_V, μ) is L-block independent.*
2. *For any μ, the pdb (λ_V, μ) is L-block 2-independent.*
3. *For any two tuples t_i, t_j, if $t_i.L \neq t_j.L$ then the Boolean expressions $\lambda(t_i)$ and $\lambda(t_j)$ do not have any common critical variables.*

Proof. Obviously (1) implies (2), and from the Lemma 5.2.4 it follows that (2) implies (3) (since $\mu(t_i) = \mu(\lambda(t_i))$ and $\mu(t_i, t_j) = \mu(\lambda(t_i) \wedge \lambda(t_j))$). For the last implication, let μ be any product probability space, and consider some m tuples $t_{i_1}, \ldots, t_{i_m}$ that have distinct values for their L attributes. Then, the Boolean expressions $\lambda(t_{i_1}), \ldots, \lambda(t_{i_m})$ depend on disjoint sets of Boolean variables, hence $\mu(t_1, \ldots, t_m) = \mu(\varphi_{i_1}, \ldots, \varphi_{i_m}) = \mu(\varphi_{i_1}) \cdots \mu(\varphi_{i_m}) = \mu(t_1) \cdots \mu(t_m)$, proving that they are independent. Thus, the three statements above are equivalent. □

Continuing the proof of Theorem 5.2.5, consider two sets of attributes L_1, L_2 such that each satisfies condition (3) in Lemma 5.2.6. Then their union, $L_1 \cup L_2$, also satisfies condition (3): indeed, if t_i and t_j are two tuples such that $t_i.(L_1 \cup L_2) \neq t_j.(L_1 \cup L_2)$, then either $t_i.L_1 \neq t_j.L_1$ or $t_i.L_2 \neq t_j.L_2$, and in either case $\lambda(t_i)$ and $\lambda(t_j)$ do not have any common critical tuples. It follows that there exists a maximal set of attributes L that satisfies condition (3), which proves the theorem. □

As an application of this result, we show how to apply it to a view V defined by an FO expression over a BID database. Let R be a relational schema, and let $PDB = (T, \mu)$ be a BID database, where T is a set of possible tuples.

Corollary 5.2.7. *For every FO view definition v over the relational schema R and for every set of possible tuples T there exists a unique maximal set of attributes L such that: for any BID database $PDB = (T, \mu)$ the probabilistic view $v(T, \mu)$ is L-block independent.*

The proof follows immediately from Theorem 5.2.5 and the observation that the view $v(PDB)$ is a lineage database.

We end this section by noting that, in general, no unique minimal K exists. For example the lineage table below has two minimal keys, $\{A\}$ and $\{B\}$:

A	B	
a_1	b_1	$X = 1 \wedge Y = 1$
a_1	b_2	$X = 1 \wedge Y = 2$
a_2	b_1	$X = 2 \wedge Y = 1$
a_2	b_2	$X = 2 \wedge Y = 2$

Therefore, any lineage database defined over this table is both A-block disjoint, and B-block disjoint, but it is not $\emptyset$-block disjoint.

5.2.3 Partial Representation of a Conjunctive View

In the previous section we have shown how to compute a partial representation for a given view (expressed in FO) and a given input BID database. We now study how to compute a partial representation given only the view expression, and not the input database. In this case we seek a partial representation (L, K) that is satisfied by the the view for *any* input BID database. This partial representation depends only on the view expression, not the data instance, and is computed through static analysis on the view expression only. Throughout this section we will restrict the view expression to be a conjunctive query.

Fix the schema R of a BID database, and let V be defined by a conjunctive query v over R. Given a BID input, PDB, we denote $V = v(PDB)$ the probabilistic table obtained by evaluating the view on the BID input. We prove in this section two results.

Theorem 5.2.8. *Fix a relational schema* R.

1. *For any conjunctive query v there exists a unique maximal set of attributes $L \subseteq Attrs(v)$ such that for any BID database PDB over the schema* R, $v(PDB)$ *is L-block independent.*

2. *The problem* "given a conjunctive query v and $L \subseteq Attrs(v)$ check whether for any BID database PDB, $v(PDB)$ is L-block independent", *is in* $\Pi_2\mathsf{P}$. *Moreover, there exists a schema* R *for which this problem is* $\Pi_2\mathsf{P}$*-hard.*

The theorem, in essence, says that there exists a unique maximal set of attributes L, and computing such a set is $\Pi_2\mathsf{P}$-complete in the size of the conjunctive query v. To obtain a good partial

representation (L, K) for the view v, we also need to compute K: finding K is equivalent to inferring functional dependencies on the output of a conjunctive view, from the key dependencies in the input schema R. This problem is well studied [1], and we will not discuss it further, but note that, in general, there may not be a unique minimal set K.

Before proving Theorem 5.2.8 we give some examples of partial representations for conjunctive views.

Example 5.2.9 1. Consider the schema $R(\underline{A})$, $S(\underline{A, B}, C, D)$, and the view:

$$v(x, y, z) \quad :- \quad R(\underline{x}), S(\underline{x, y}, z, u)$$

Denote $V(X, Y, Z)$ the schema of the materialized view. A partial representation for V is (X, XY). To see that V is X-block independent, notice that if two tuples in V differ on their X attribute, then the lineage of the two tuples depends on disjoint sets of input tuples in R and S. To see that V is XY-block disjoint, it suffices to see that the functional dependency $XY \rightarrow Z$ holds in V (because it holds in S). Thus, (X, XY) is a partial representation for V, and one can see that it is the best possible (that is, we cannot increase X nor decrease XY).

2. Consider the schema $R(\underline{A, B}, C)$, $S(\underline{A, C}, B)$ and the view:

$$v(x, y, z) \quad :- \quad R(\underline{x, y}, z), S(\underline{x, z}, y)$$

Here V is X-block independent. In addition, V is both XY-block disjoint and XZ-block disjoint: but it is not X-block disjoint. There are two "best" partial representations: (X, XY) and (X, XZ).

In the remainder of this section we will prove Theorem 5.2.8, and start with Part 1. Fix a BID probabilistic database PDB. Then $V = v(PDB)$ is a lineage database: by Theorem 5.2.5 there exists a unique maximal set of attributes L_{PDB} such that V is L-block independent (for any choice of the probability function in PDB). Then the set of attributes $\bigcap_{PDB} L_{PDB}$ is the unique, maximal set of attributes claimed by the theorem.

Before proving part 2 of Theorem 5.2.8, we need to review the notion of a critical tuple for a Boolean query q.

Definition 5.2.10. *A ground tuple t is called* critical *for a Boolean query q if there exists a (conventional) database instance I s.t. $q(I) \neq q(I \cup \{t\})$.*

A tuple is critical if it makes a difference for the query. For a simple illustration, consider the Boolean query $q :- R(x, x), S(a, x, y)$, where a is a constant. Then $R(b, b)$ (for some constant b) is a critical tuple because q is false on the instance $I = \{S(a, b, c)\}$ but true on the instance $\{R(b, b), S(a, b, c)\}$. On the other hand $R(b, c)$ is not a critical tuple. In general, if the query q is a conjunctive query, then any critical tuple must be the ground instantiation of a subgoal. The converse is not true as the following example shows due to Miklau and Suciu shows [119] : $q :- R(x, y, z, z, u), R(x, x, x, y, y)$. The tuple $t = R(a, a, b, b, c)$, which is a ground instantiation of the first subgoal, is not a critical tuple. Indeed, if q is true on $I \cup \{t\}$, then only the first subgoal can be mapped to t, and therefore the second subgoal is mapped to the ground tuple $R(a, a, a, a, a)$, which must be in I: but then q is also true on I, hence t is not critical. We review here the main results Miklau and Suciu [119]. While these results were shown for a relational schema R where the key of each relation is the set of all its attributes (in other words, any BID database over schema R is a tuple-independent probabilistic database), they extend immediately to BID probabilistic databases (the main step of the extension consists of Lemma 5.2.4).

Theorem 5.2.11. *[119] Fix a relational schema* R.

1. *Let q, q' be two Boolean conjunctive queries over the schema* R. *Then the following two statements are equivalent: (a) q and q' have no common critical tuples, (b) for any BID probabilistic database over the schema* R*, q and q' are independent events.*

2. *The problem* "given two Boolean queries q, q', check whether they have no common critical tuples" *is in* $\Pi_2\mathsf{P}$.

3. *There exists a schema* R *such that the problem* "given a Boolean query q and a ground tuple t, check whether t is not a critical tuple for q", *is* $\Pi_2\mathsf{P}$*-hard.*

We now prove part 2 of Theorem 5.2.8. Given a set of attributes L, the following two conditions are equivalent: (a) for any input PDB, $v(PDB)$ is L-block independent, and (b) for any two distinct grounded $|L|$-tuples $\bar{a}$, $\bar{b}$, the two Boolean queries $v(\bar{a})$ and $v(\bar{b})$ have no common critical tuples. (Here $v(\bar{a})$ denotes the Boolean query where the head variables corresponding to the L attributes are substituted with $\bar{a}$, while the rest of the head variables are existentially quantified). The equivalence between (a) and (b) follows from Lemma 5.2.6 (2) and Theorem 5.2.11 (1). Membership in $\Pi_2\mathsf{P}$ follows now from property (b) and Theorem 5.2.11 (2).

To prove hardness for $\Pi_2\mathsf{P}$, we use Theorem 5.2.11 (3): we reduce the problem "given a query q and a tuple t check whether t is not a critical tuple for q" to the problem (b) above. Let R be the vocabulary for q and t, and let the ground tuple t be $T(a_1, \ldots, a_k)$. Construct a new vocabulary R' obtained from R by adding two new attributes to each relation name: that is, if $R(A_1, \ldots, A_m)$ is a relation name in R, then $R'(U, V, A_1, \ldots, A_m)$ is a relation name in R'. Let U, V be two variables. Denote $q'(U, V)$ the query obtained from q by replacing every subgoal $R(\ldots)$ in q with $R'(U, V, \ldots)$ (thus, the variables U, V will occur in all subgoals of $q'(U, V)$), and define the following view:

$$v(U, V) \quad :- \quad q'(U, V), T(V, U, a_1, \ldots, a_k)$$

We show that v is UV-block independent iff t is not a critical tuple for q. For that, we consider two distinct constant tuples (u_1, v_1) and (u_2, v_2) and examine whether the Boolean queries $v(u_1, v_1)$ and $v(u_2, v_2)$ are independent, or, equivalently, have no common critical tuples. All critical tuples of $v(u_1, v_1)$ must have the form $R(u_1, v_1, \ldots)$, or $T(v_1, u_1, \ldots)$; in other words, they must start with the constants u_1, v_1 or with v_1, u_1. Similarly for $v(u_2, v_2)$; hence, if $\{u_1, v_1\} \neq \{u_2, v_2\}$ then the queries $v(u_1, v_1)$ and $v(u_2, v_2)$ have no common critical tuples. The only case when they could have common critical tuples is when $u_1 = v_2$ and $u_2 = v_1$ (since $(u_1, v_1) \neq (u_2, v_2)$), and in that case they have a common critical tuple iff $T(u_1, v_1, a_1, \ldots, a_k)$ is a critical tuple for $q'(u_1, v_1)$, and this happens iff $T(a_1, \ldots, a_k)$ is a critical tuple for q. This completes the hardness proof.

5.2.4 Querying Partially Represented Views

We have shown how to compute a "best" partial representation (L, K) for a materialized view V: tuples with distinct values for L are independent, while tuples that agree on K are disjoint. All other pairs of tuples, which we call *intertwined*, have unspecified correlations. In this section we study the problem of deciding whether a query q can be answered by using only the marginal probabilities in V: we say that q is *well-defined* in terms of the view and its partial representation. Intuitively, q does not look at pairs of intertwined tuples. This problem is complementary to the query answering using views problem [85]: there, we are given a query q over a conventional database and a set of views, and we want to check if q can be rewritten into an equivalent query q' that uses the views. We assume that the rewriting has already been done, thus q already mentions the view(s).

We illustrate with an example:

Example 5.2.12 Let $V(A, B, C)$ have the following partial representation: $L = A$, $K = AB$. Consider the following queries:

$$\begin{aligned} q_1 &:\!- \quad V(a, y, z) \\ q_2 &:\!- \quad V(x, b, y) \\ q_3 &:\!- \quad V(x, y, c) \end{aligned}$$

Here x, y, z are variables, and a, b, c are constants. For example, the view could be that from Example 5.2.9 (1), $v(x, y, z) :\!- R(\underline{x}), S(\underline{x, y}, z, u)$, and the query q_1 could be $q_1 :\!- R(\underline{a}), S(\underline{a, y}, z, u)$: after rewriting q_1 in terms of the view, we obtain the equivalent expression $q_1 :\!- V(a, y, z)$.

We argue that q_2 is well-defined, while q_1, q_3 are not. Consider first q_2. Its value depends only on the tuples of the form (a_i, b, c_j), where the constants a_i and c_j range over the active domain, and b is the fixed constant occurring in the query. Partition these tuples by a_i. For each $i = 1, 2, \ldots$, any two tuples in the set defined by a_i are disjoint (because V satisfies the partial representation (A, AB), and any two tuples in the same group agree on both A and B): thus, the Boolean query $\exists z.V(a_i, b, z)$ is a disjunction of the exclusive events $V(a_i, b, c_j)$, and therefore its probability is the sum of the probabilities $\mu(V(a_i, b, c_j))$. Moreover, the set of events $\{\exists z.V(a_1, b, z), \exists z.V(a_2, b, z), \ldots\}$,

is independent, which allows us to compute the probability of the query q_2, since it is the disjunction of these independent events: $q_2 \equiv \exists x.\exists z.V(x, b, z)$. Thus, assuming that the view V satisfies the partial representation (A, AB), the probability of q_2 depends only on the marginal tuples probabilities in the view V.

In contrast, neither q_1 nor q_2 are well-defined. To see this, suppose that the view has exactly two tuples, $t_1 = (a, b_1, c)$ and $t_2 = (a, b_2, c)$. These tuples are intertwined, i.e.the probability $\mu(t_1, t_2)$ is unknown. Further, $\mu(q_1) = \mu(q_3) = \mu(t_1 \vee t_2) = \mu(t_1) + \mu(t_2) - \mu(t_1 t_2)$: $\mu(t_1)$ and $\mu(t_2)$ are well defined in the view, but $\mu(t_1, t_2)$ is unknown. So, neither q_1 nor q_3 is well-defined.

In this section we consider Boolean, monotone queries q, which includes conjunctive queries. We assume that q is over a single view V, and mentions no other relations. This is not too restrictive, since q may have self-joins over V, or unions (since we allow arbitrary monotone queries). It is straightforward to extend our results to a query expressed over multiple views $V_1, V_2, \ldots$, each with its own partial representation, assuming that all views are independent.

Definition 5.2.13. *Let V be a view with a partial representation (L, K), and let q be a monotone Boolean query over the single relation name V. We say that q is well-defined given the partial representation (L, K), if for any two probabilistic relations $PV = (\mathcal{W}, \mu)$ and $PV' = (W', \mu')$ that satisfy the partial representation (L, K), and that have identical tuple probabilities*[2]*, the following holds: $\mu(q) = \mu'(q)$.*

Thus, q is well defined iff $\mu(q)$ depends only on the marginal tuple probabilities $\mu(t)$ (which we know), and not on the entire distribution μ (which we don't know). We will give now a necessary and sufficient condition for q to be well defined, but first we need to introduce two notions: intertwined tuples, and a set of critical tuples.

Definition 5.2.14. *Let (L, K) be a partial representation of a view V. Let t, t' be two ground tuples of the same arity as V. We call t,t'* intertwined *if $t.L = t'.L$ and $t.K \neq t'.K$.*

Next, we generalize a critical tuple (see Definition 5.2.10) to a set of critical tuples. Let $Inst = \mathcal{P}(Tup)$ be the set of (conventional) database instances over the set of ground tuples Tup. To each Boolean query q we associate the real-valued function $f_q : Inst \to \mathbb{R}$:

[2] $\forall t\ \mu(t) = \mu'(t)$

$$f_q(I) \quad = \quad \begin{cases} 1 & \text{if } q(I) \text{ is true} \\ 0 & \text{if } q(I) \text{ is false} \end{cases}$$

Definition 5.2.15. *Let $f : Inst \to \mathbb{R}$ be a real-valued function on instances. The* differential *of f w.r.t. a set of tuples $S \subseteq Tup$ is the real-valued function $\Delta_S f : Inst \to \mathbb{R}$ defined as follows:*

$$\begin{aligned} \Delta_\emptyset f(I) &= f(I) \\ \Delta_{\{t\} \cup S} f(I) &= \Delta_S f(I) - \Delta_S(f(I - \{t\})) \ \ \textit{if } t \notin S \end{aligned}$$

Definition 5.2.16. *A set of tuples $C \subseteq Tup$ is critical for f if there exists an instance I s.t. $\Delta_C f(I) \neq 0$. A set of tuples C is critical for a Boolean query q if it is critical for f_q.*

We can now state the main result in this section:

Theorem 5.2.17. *Let V be a view with a partial representation (L, K).*

1. *A monotone Boolean query q over V is well defined iff for any two intertwined tuples t, t' the set $\{t, t'\}$ is not critical for q.*

2. *The problem* "given a Boolean conjunctive query q over the view V, decide whether q is well defined" *is in Π_2P. Moreover, there exists a view V and partial representation (L, K) for which this problem is Π_2P hard.*

Thus, in order to evaluate a query q using a view V with partial representation (L, K) one proceeds as follows. First, we check if q is well defined, by checking if it has no pair of intertwined, critical tuples: this is a Π_2P-complete problem in the size of the query q. Second, if this holds, then we evaluate q over V by assuming V is a BID table with key attributes L; or, alternatively, we may assume a BID table with key attributes K. The well-definedness condition ensures that we obtain the same answer over any of these two BID tables as over the view V.

In the rest of the section we prove Theorem 5.2.17, and for that we give a definition, and three lemmas.

Definition 5.2.18. *Let $T \subseteq Tup$ be a set of tuples. The* restriction *of a real-valued function f to T is: $f^T(I) = f(I \cap T)$. Similarly, the restriction of a Boolean query q to T is: $q^T(I) = q(I \cap T)$.*

The first lemma establishes some simple properties of critical sets of tuples. Note that a set of tuples C is not critical for f iff $\Delta_C f = 0$, meaning $\forall I : \Delta_C f(I) = 0$.

Lemma 5.2.19. *Let C be a set of tuples and suppose $\Delta_C f = 0$. Then:*

1. *For any set of tuples $D \supseteq C$, $\Delta_D f = 0$.*
2. *For any set of tuples S, $\Delta_C \Delta_S f = 0$.*
3. *For any set of tuples T, $\Delta_C f^T = 0$.*

Proof. (1): we show that $\Delta_D f(I) = 0$ by induction on the size of D. If $D = C$ then it follows from the assumption that $\Delta_C f = 0$. We show this for $D \cup \{t\}$, where $t \notin D$: $\Delta_{D\cup\{t\}}(I) = \Delta_D(I) - \Delta_D(I - \{t\}) = 0 - 0 = 0$. (2): $\Delta_C \Delta_S f(I) = \Delta_{S\cup C} f(I)$, and the latter is 0 by the previous claim. (3): $\Delta_C f^T(I) = \Delta_C f(I \cap T) = 0$, because $\Delta_C f = 0$. □

The second lemma gives a series expansion for any real-valued function $f : Inst \to \mathbb{R}$, in terms of its differentials.

Lemma 5.2.20. *For any set of tuples $T \subseteq Tup$:*

$$f = \sum_{S \subseteq T} \Delta_S f^{Tup-(T-S)} \tag{5.3}$$

As a consequence:

$$f = \sum_{S \subseteq Tup} \Delta_S f^S \tag{5.4}$$

Equation (5.3) can be written equivalently as $f(I) = \sum_{S\subseteq T} \Delta_S f(I - (T - S))$. For example, by setting $T = \{t\}$ or $T = \{t_1, t_2\}$ in Eq.(5.3) we obtain:

$$\begin{aligned} f(I) &= f(I - \{t\}) + \Delta_t f(I) \\ f(I) &= f(I - \{t_1, t_2\}) + \Delta_{t_1} f(I - \{t_2\}) + \Delta_{t_2} f(I - \{t_1\}) + \Delta_{t_1,t_2} f(I) \end{aligned}$$

Proof. (Sketch) We prove (5.3) by induction on the size of the set T. The first example above shows the base case. Assuming $s \notin T$, we can split the sum over $S \subseteq T \cup \{t\}$ into to sums: one iterating over S where $S \subseteq T$ and the other iterating over $S \cup \{t\}$ where $S \subseteq T$:

$$\begin{aligned}
&\sum_{S \subseteq T \cup \{t\}} \Delta_S f^{Tup-(T\cup\{t\}-S)}(I) = \\
&= \sum_{S \subseteq T} \Delta_S f^{Tup-(T\cup\{t\}-S)}(I) + \sum_{S \subseteq T} \Delta_{S\cup\{t\}} f^{Tup-(T-S)}(I) \\
&= \sum_{S \subseteq T} \Delta_S f^{Tup-(T\cup\{t\}-S)}(I) + \sum_{S \subseteq T} \Delta_S f^{Tup-(T-S)}(I) - \sum_{S \subseteq T} \Delta_S f^{Tup-(T-S)}(I - \{t\}) \\
&= \sum_{S \subseteq T} \Delta_S f^{Tup-(T-S)}(I) = f(I)
\end{aligned}$$

The last identity holds because $f^{Tup-(T\cup\{t\}-S)}(I) = f^{Tup-(T-S)}(I - \{t\})$. This completes the proof of (5.3). To prove (5.4) we set $T = Tup$ in (5.3). □

For the third lemma, we fix the partial representation (L, K) of the view.

Definition 5.2.21. *A set of ground tuples T is non-intertwined, or NIT, if $\forall t, t' \in T$, t and t' are not intertwined. In other words: $\forall t, t' \in T$, either $t.L \neq t'.L$ or $t.K = t'.K$.*

Lemma 5.2.22. *Let (L, K) be a partial representation of a view V, and let q be a monotone Boolean query over V. Assume q has no critical pairs of intertwined tuples, and let T be a NIT set of tuples. Then q^T is well defined given the partial representation (L, K) of the view V.*

Proof. A *minterm* for q^T is a minimal instance J s.t. $q^T(J)$ is true; that is, J is a set of tuples s.t. $q^T(J)$ is true and for all $J' \subseteq J$, if $q^T(J')$ is true then $J = J'$. Denote M the set of all minterms for q^T. Obviously, each minterm for q^T is a subset of T. Since q^T is monotone (because q is monotone), it is uniquely determined by M:

$$q^T(I) \quad = \quad \bigvee_{J \in \mathsf{M}} (J \subseteq I)$$

In other words, q^T is true on an instance I iff the set of tuples I contains a minterm J. Denote r^J the

Boolean query $r^J(I) = (J \subseteq I)$, we apply the inclusion-exclusion formula to derive:

$$\mu(q^T) \quad = \quad \mu(\bigvee_{J \in \mathsf{M}} r^J) = \sum_{N \subseteq \mathsf{M}, N \neq \emptyset} (-1)^{|N|} \mu(r^{\bigcup N})$$

Finally, we observe that for each $N \subseteq \mathsf{M}$, the expression $\mu(r^{\bigcup N})$ is well defined. Indeed, the set $J = \bigcup N$ is the union of minterms in N, thus it is a subset of T, hence it is a NIT set. If $J = \{t_1, t_2, \ldots\}$, the query r^J simply checks for the presence of all tuples $t_1, t_2, \ldots$; in more familiar notation $\mu(r^J) = \mu(t_1 t_2 \cdots)$. If the set J contains two disjoint tuples ($t_i.K = t_j.K$) then $\mu(t_1 t_2 \cdots) = 0$. Otherwise, it contains only independent tuples ($t_i.L \neq t_j.L$), hence $\mu(t_1 t_2 \cdots) = \mu(t_1)\mu(t_2) \cdots$ In either cases it is well-defined and, hence, so is $\mu(q^T)$. □

We now prove Theorem 5.2.17

Proof. Part 1: We start with the "only if" direction. Let q be well defined, and assume it has a pair t, t' of intertwined, critical tuples. By definition there exists an instance I s.t. $f_q(I) - f_q(I - \{t\}) - f_q(I - \{t'\}) + f_q(I - \{t, t'\}) \neq 0$. Since q is monotone it follows that f_q is monotone, hence $q(I) = \mathit{true}$, $q(I - \{t, t'\}) = \mathit{false}$, and either $q(I - \{t\}) = q(I - \{t'\}) = \mathit{false}$ or $q(I - \{t\}) = q(I - \{t'\}) = \mathit{true}$. Without loss of generality, assume that $q(I - \{t\}) = q(I - \{t'\}) = \mathit{false}$. Then we define two probabilistic databases $PV = (\mathcal{W}, \mu)$ and $PV' = (\mathcal{W}, \mu')$ as follows. Each has four possible worlds: $I, I - \{t\}, I - \{t'\}, I - \{t, t'\}$. In PV these worlds are assigned probability $\mu = (0.5, 0, 0, 0.5)$, respectively; here, t_1 and t_2 are positively correlated. In PV', all worlds are assigned probability 0.25 i.e.the two tuples are independent. Observe that in both cases, the marginal probability of any tuple is the same, $\mu(t) = \mu(t') = 0.5$ and all other tuples have probability 1. Then we have $\mu(q) = 0.5$ and $\mu'(q) = 0.25$, contradicting the assumption that q is well-defined.

Next we prove the "if" part, so assume q has no pair of intertwined, critical tuples. The basic plan is this. Suppose an instance I contains two intertwined tuples t, t' (hence we don't know their correlations). Write $q(I) = q(I - \{t, t'\}) + \Delta_t q(I - \{t'\}) + \Delta_{t'} q(I - \{t\})$ (because $\Delta_{t,t'} q = 0$). Thus, we can "remove" t or t' or both from I and get a definition of q on a smaller instance, and by repeating this process we can eliminate all intertwined tuples from I, then we apply Lemma 5.2.22

Formally, let q be a monotone Boolean query that has no critical pair of intertwined tuples for (L, K). Let $PV = (\mathcal{W}, \mu)$ be a probabilistic database that satisfies the partial representation (L, K),

and let Tup be the set of possible tuples in PV. We expand f_q using Lemma 5.2.20:

$$\begin{aligned} f_q &= \sum_{T \subseteq Tup} \Delta_T f_q^T \\ &= \sum_{T \subseteq Tup : T \text{ is NIT}} \Delta_T f_q^T \\ &= \sum_{T \subseteq Tup : T \text{ is NIT}} \sum_{S \subseteq T} (-1)^{|S|} f_q^{T-S} \end{aligned} \tag{5.5}$$

The first line is Eq.(5.4) in Lemma 5.2.20. To show the second line, we start from the fact that $\Delta_{\{t,t'\}} f_q = 0$ when t, t' are intertwined tuples, because we assumed that q has no critical pair of intertwined tuples. Then, every set T that is not NIT can be written as $T = \{t, t'\} \cup T'$ where t, t' are two intertwined tuples. We apply Lemma 5.2.19 twice: $\Delta_{\{t,t'\}} f_q = 0$ implies $\Delta_{\{t,t'\}} f_q^T = 0$, which further implies $\Delta_T f_q^T = 0$. Thus, the only terms in the first line that are non-zero are those that correspond to sets T that are NIT: this is what the second line says. Finally, the last line is the direct definition of Δ_T.

Next we apply the expectation on both sides of Eq.(5.5), and use the linearity of expectation plus $\mu(q) = E[f_q]$:

$$\begin{aligned} \mu(q) = E[f_q] &= \sum_{T \subseteq Tup : T \text{ is NIT}} \sum_{S \subseteq T} (-1)^{|S|} E[f_q^{T-S}] \\ &= \sum_{T \subseteq Tup : T \text{ is NIT}} \sum_{S \subseteq T} (-1)^{|S|} \mu(q^{T-S}) \end{aligned}$$

The claim of the theorem follows from that fact that by Lemma 5.2.22 each expression $\mu(q^{T-S})$ is well defined.

We now prove Part 2, by providing a reduction from the problem *"given a conjunctive query q and a tuple t check whether t is critical for q"*. Assume w.l.o.g. that the query and the tuple are over a vocabulary with a single relation symbol (namely V). If not, we rename relation symbols by padding them so that they have the same arities, then adding constants: for example if the vocabulary is $R_1(A, B), R_2(C, D, E, F), R_3(G, H, K)$, then we will rewrite both the query and the tuple using a

single relation symbol V of arity 5 (the largest of the three arities plus one), and replace $R_1(x, y)$ with $V(x, y, a, a)$, replace $R_2(x, y, z, u)$ with $V(x, y, z, u, b)$, and replace $R_3(x, y, z)$ with $V(x, y, z, c, c)$, where a, b, c are fixed constants.

Thus, we have a query q and a ground tuple t, over a single relation symbol V of arity k, in particular $t = V(c_1, \ldots, c_k) = V(\bar{c})$. We want to check whether t is a critical tuple for q. We reduce this problem to the problem of checking whether some query q' is well defined over some view V'; the new view will have arity $k + 1$. Fix two distinct constants a, b. The new query q' is obtained from q by replacing every subgoal $V(x, y, \ldots)$ with $V'(a, x, y, \ldots)$, and by adding the constant subgoal $V(b, c_1, \ldots, c_k)$. Thus, queries q and q' look like this:

$$\begin{aligned} q &= V(\bar{x_1}), V(\bar{x_2}), \ldots, V(\bar{x_m}) \\ q' &= V'(a, \bar{x_1}), V'(a, \bar{x_2}), \ldots, V'(a, \bar{x_m}), V'(b, \bar{c}) \end{aligned}$$

Consider the partial representation (L, K) for V', where $L = Attr(V)$ and $K = Attr(V')$. Recall that q' is well defined over this partial representation, iff it has no pairs of intertwined, critical tuples. Thus, to prove the hardness claim it suffices to show that the following two statements are equivalent:

1. There exists two intertwined, critical tuples for q'.

2. t is a critical tuple for q.

We start by showing 1 implies 2. Two tuples t_1, t_2 are intertwined iff they agree on the L attributes, $t_1.L = t_2.L$, and disagree on the K attributes, hence $t_1.A \neq t_2.A$, where $A = Attrs(V') - L$ is the extra attribute that was added to V'. On the other hand, if the set $\{t_1, t_2\}$ is also critical for q', then $t_1.A = a$ and $t_2.A = b$ (since q' only inspects tuples that have $A = a$ or $A = b$), and, moreover, $t_1.L = t_2.L = \bar{c}$ (since the only tuple with $A = b$ that is critical for q' is $V'(b, \bar{c})$). Let I' be an instance that witnesses the fact that $\{t_1, t_2\}$ is doubly critical:

$$\begin{aligned} 0 \neq \Delta_{t_1,t_2} f_{q'}(I') &= f_{q'}(I') - f_{q'}(I' - \{t_1\}) - f_{q'}(I' - \{t_2\}) + f_{q'}(I' - \{t_1, t_2\}) \\ &= 1 - f_{q'}(I' - \{t_1\}) - 0 + 0 \end{aligned}$$

We used the fact that $f_{q'}(I') = 1$ (otherwise, if $f_{q'}(I') = 0$ then $\Delta_{t_1,t_2}(f_{q'}(I')) = 0$), which implies that

$t_2 = V'(b, \overline{c}) \in I'$. Thus, we have $q'(I') = \mathit{true}$ and $q'(I' - \{t_1\}) = \mathit{false}$. We construct from here an instance I such that $q(I) = \mathit{true}$ and $q(I - \{t\}) = \mathit{false}$: indeed, take $I = \{V(\overline{d}) \mid V(a, \overline{d}) \in I'\}$, it obviously satisfies this property. Thus, I is a witness for t being a critical tuple for q.

To prove 2 implies 1 we use the same argument, in reverse. We start with an instance I such that $q(I) = \mathit{true}$, $q(I - \{t\}) = \mathit{false}$, and define $I' = \{V(a, \overline{d}) \mid V(\overline{d}) \in I\} \cup \{V(b, \overline{c})\}$. It is easy to check that $\Delta_{t_1,t_2} f_{q'}(I') \neq 0$. This completes the proof. □

5.3 Practical Query Answering using Probabilistic Views

In this section, we consider practical enhancements to the theory of probabilsitic views. Concretely, the complexity of the decision problems in the previous section is high. In this section, we give efficient approximation algorithms for both problems that are sound (but not complete). For a restricted class of queries, we are able to prove that our PTIME algorithms are complete. We then discuss further practical enhancements to our theory, such as coping with dependencies.

5.3.1 Practical Algorithm for Representability

Checking L-block independence is intractable, and so we give a polynomial time approximation that is sound, i.e. it says a view is representable only if it is representable, but not complete, the test may say that a view is not L-block independent when in fact it is. The central notion is a ***k*-collision**, which intuitively says there are two output tuples which may depend on input tuples that are not independent (i.e. the same tuple or disjoint).

To make this precise, let $q = g_1, \ldots, g_n$ be a conjunctive query where k_i denotes the tuple of variables and constants in the possible worlds key position of the subgoal g_i. For example,

$$q = R(\underline{x, x, y}; z), R(\underline{x, y, y}; u), S(\underline{x, \text{`a'}}; z)$$

then $k_1 = (x, x, y)$, $k_2 = (x, y, y)$, and $k_3 = (x, \text{`a'})$. We say that a valuation $v : \mathbf{var}(q) \to \mathbb{D}$ is *disjoint aware* if for $i, j = 1, \ldots, n$, we have $v(k_i) = v(k_j)$ and $\mathbf{pred}(g_i) = \mathbf{pred}(g_j)$ then $v(g_i) = v(g_j)$. The intution is that this is a valuation which is consistent with the possible worlds key constraints. Above, $v(x, y, z, u) = (a, b, c, d)$ is disjoint aware, while $v(x, y, z, u) = (a, a, c, d)$ is not: there is no possible world that contains both of the tuples $R(\underline{a, a, a}; c)$ and $R(\underline{a, a, a}; d)$, since this would violate

the possible worlds key constraint.

Definition 5.3.1. *A* $\boldsymbol{L}$-collision *for a view*

$$V^p(L, U) :- g_1, \ldots, g_n$$

is a pair of disjoint aware valuations (v, w) *such that* $v(\boldsymbol{L}) \neq w(\boldsymbol{L})$ *but there exists* i, j *such that* g_i *that is probabilistic,* $\mathbf{pred}(g_i) = \mathbf{pred}(g_j)$ *and* $v(\boldsymbol{k}_i) = w(\boldsymbol{k}_j)$.

The utility of this definition is that we will show that if we cannot find an L-collision, then we are sure that the output of the view will be L-independent.

Example 5.3.2 Consider $V_2^p(C)$ in Eq. (5.2), if we unify any pair of probabilistic subgoals, we are forced to unify the head, c. This means that a collision is never possible and we conclude that V_2^p is C-block independent. Notice that we can unify the S subgoal for distinct values of c, since S is deterministic, this is not a collision. In $V_1^p(c, r)$ Eq. (5.1), the following pair (v, w), $v(c, r, d) =$ ('TD', 'D.Lounge','Crab Cakes') and $w(c, r, d) =$ ('TD', 'P.Kitchen', 'Crab Cakes'), is a collision because $v(c, r) \neq w(c, r)$ and we have unified the keys of the $\mathtt{R}^p$ subgoal. Since there are no repeated probabilistic subgoals, we are sure that V_1^p is not CR-block independent.

We need the following proposition:

Proposition 5.3.3. *If* $V(L, U)$ *is a view on a BID database without an* L*-collision. Then for any tuples* $s, t \in V$ *such that* $s[L] \neq t[L]$ *the sets of variables in* $\lambda(s)$ *and* $\lambda(t)$ *are disjoint.*

Proof. This is immediate from the definition. □

Lemma 5.3.4. *If a view* $v(L, U)$ *on a BID table does not have an* L*-collision, then* v *is* L*-independent. Moreover, if* v *does not contain repeated predicates then the converse holds as well.*

Proof. If there does not exist an L-collision, then for any tuples s, t such that $s[L] \neq t[L]$ that appear in the view V, then $\lambda(s)$ and $\lambda(t)$ have no common variables. Hence, no critiacl tuple could be shared between them and so the view is L-independent. If there are no repeated predicates in v, then we observe that *every tuple* in the image of any valuation is critical. Let v, w be the valuations provided by the definition and g_i, g_j be subgoals such that $v(k_i) = w(k_j)$ and $v(L) \neq w(L)$. Then, this implies the tuples $v(g_i)$ and $v(g_j)$ are pair critical. □

The last thing we must observe is that there is an easy, complete test for L-collisions:

- Create two copies of the query $V(L, U)$ and put them into one query $VV(L, U, L', U') = V(L, U), V'(L', U')$.
- For each pair of subgoals $g_i \in V$ and $g' \in V'$, unify them in VV and then construct the most general unifier [159] that respects the key constraints in the BID instances. If we can construct a such that $L \neq L'$, then reject the query: this unifier encodes an L-collision. To see this, v (resp. w) is the unifier restricted to the variables in V (resp. V'). If not, then there is no L-collision.

The second step can be done using the Chase for functioanl dependencies, which is done in polynomial time [1]. Thus, we can find L-collisions in polynomial time. Summarizing, we have the following theorem:

Theorem 5.3.5. *Given a conjunctive view $V(L, U)$ on a BID instance. The previous algorithm gives a polynomial-time, sound test for testing L-independence. Moreover, if V does not contain self-joins, then the test is complete.*

5.3.2 Practical Test for Uniqueness

We now give a test in a similar spirit for uniqueness. We say that a pair of disjoint-aware valuations v, w is *compatible* if for $i, j = 1, \ldots, n$, whenever we have $v(k_i) = w(k_j)$ and $\mathbf{pred}(g_i) = \mathbf{pred}(g_j)$, then it must be that $v(g_i) = w(g_j)$. Intuitively, if a pair of valuations is compatible it means that there is some *possible* world W where both of these valuations can map the query simultaneously, i.e. $v(q) \cup w(q) \subseteq W$.

Definition 5.3.6. *Given a schema with a single view V that has a partial representation (L, K), an* intertwined collision *for a query $Q(H)$ is a pair of compatible valuations (v, w) such that $v(H) = w(H)$ and there exists a pair of subgoals, (g_i, g_j), such that $v(g_i)$ and $v(g_j)$ are intertwined.*

The intution of an intertwined collision is that there are two derivations for a tuple in the output of Q that depend on a pair of intertwined tuples, i.e., tuples whose correlation is not specified by the partially represent view.

Example 5.3.7 In V_1, $K_I = \{C\}$ and $D = \{R\}$. An intertwined collision for Q_1 is $v(c,r)$ = ('TD', 'D.Lounge') and $w(c,r)$ = ('TD', 'P.Kitchen'), thus Q's value is not uniquely defined. On the other hand, in Q_2, trivially there is no intertwined collision and so Q_2's value is uniquely defined.

The algorithm to find an intertwined collision is a straightforward extension of finding a K-collision. The key difference is that we use the Chase to ensure that the valuations we find are compatible, not individually disjoint aware.

Theorem 5.3.8. *If no intertwined collisions exist for a conjunctive query Q, then its value is uniquely defined. If the partially representable view symbol V^p is not repeated, this test is complete. The test can be implemented in* PTIME.

Proof. First consider the soundness argument in the special case of a Boolean query $Q()$. We show that if there exists a critical intertwined pair (s,t) for Q, then there must be an intertwined collision. Let I be the instance provided Definition 5.2.14. Suppose, $I - \{s\} \models Q()$. Since $I - \{s,t\} \not\models Q()$, the image of any valuation v that witnesses $I - \{s\} \models Q()$ must contain t. By symmetry, the image of any valuation that witnesses $I - \{t\} \models Q()$ must contain w. It is easy to see that (v,w) is compatible and hence (v,w) is an intertwined collision. If $I - \{s\} \not\models Q()$ then there is a single valuation v which uses both s,t. Thus, (v,v) is an intertwined collision. It is straightforward to extend to the non-Boolean case. To see completeness, we observe that that *every tuple* in the image of any valuation is critical. Hence an intertwined collision finds a pair of intertwined tuples that are critical. We then apply Theorem 5.2.17. □

5.4 Experiments

In this section we answer three main questions: To what extent do representable and partially representable views occur in real and synthetic data sets? How much do probabilistic materialized views help query processing? How expensive are our proposed algorithms for finding representable views?

5.4.1 Experimental Setup

Data Description We experimented with a variety of real and synthetic data sets including: a database from iLike.com [40], the Northwind database (NW) [41], the Adventure Works Database

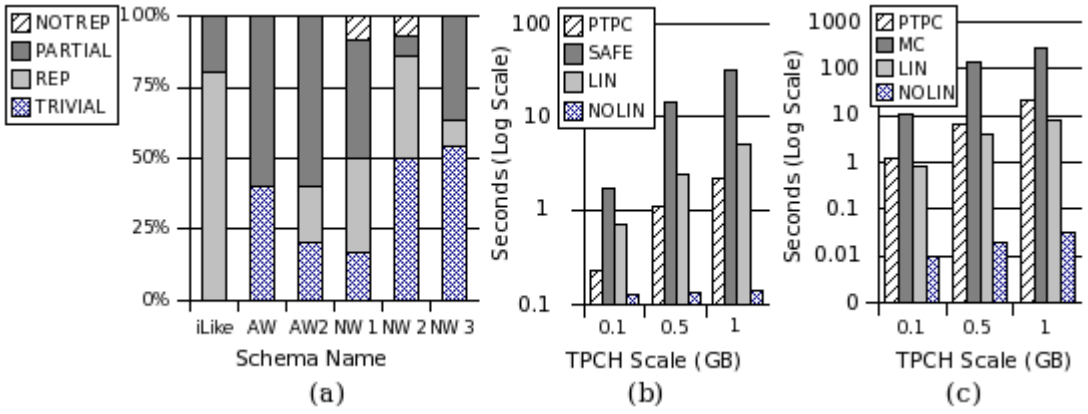


Figure 5.2: (a) Percentage by workload that are representable, non-trivially partially representable or not representable. We see that almost all views have some non-trivial partial representation. (b) Running times for Query 10 which is safe. (c) Retrieval times for Query 5 which is not safe. Performance data is TPC-H (0.1, 0.5, 1G) data sets. All running times in seconds and on logarithmic scale.

from SQL Server 2005 (AW) [42] and the TPC-H/R benchmark (TPCH) [43, 44]. We manually created several probabilistic schemata based on the Adventure Works [42], Northwind [41] and TPC-H data which are described in Fig. 5.3.

Queries and Views We interpreted all queries and views with scalar aggregation as probabilistic existence operators (i.e. computing the probability a tuple is present). iLike, Northwind and Adventure Works had predefined views as part of the schema. We created materialized views for TPC-H using an exhaustive procedure to find all subqueries that were representable, did not contain cross products and joined at least two probabilistic relations.

Real data: iLike.com We were given query logs and the relational schema of iLike.com, which is interesting for three reasons: It is a real company, a core activity of iLike is manipulating uncertain data (e.g. similarity scores) and the schema contains materialized views. iLike's data, though not natively probabilistic, is easily mapped to a BID representation. The schema contains over 200 tables of which a handful contain uncertain information. The workload trace contains over 7 million queries of which more than 100,000 manipulated uncertain information contained in 5

Schema	Tables (w/**P**)
AW	18 (6)
AW2	18 (3)
NW1	16 (2)
NW2	16 (5)
NW3	16 (4)
TPC-H	8 (5)

(a)

Size (w/**P**)	Tuples (w/**P**)
0.1 (440M)	3.3M (2.4M)
0.5 (2.1G)	16M (11.6M)
1.0 (4.2G)	32M (23.2M)

(b)

Figure 5.3: Schema and TPC Data statistics. (a) Number of tables referenced by at least one view and number of probabilistic tables (i.e. with attribute **P**). (b) Size and (w/**P**) are in Gb. The number of deterministic and probabilistic tuples is in millions.

views. Of these 100,000 queries, we identified less than 10 query types which ranged from simple selections to complicated many way joins.

Performance Data All performance experiments use the TPC-H data set with a probabilistic schema containing uncertainty in the `part`, `orders`, `customer`, `supplier` and `lineitem` tables. We used the TPC-H tool `dbgen` to generate relational data. The data in each table marked as probabilistic was then transformed by uniformly at random injecting additional tuples such that each key value was expected to occur 2.5 times. We allowed for *entity uncertainty*, that is, the sum of probabilities for a possible worlds key may be less than 1.

System Details Our experimental machine was a Windows Server 2003 machine running SQL Server 2005 with 4GB of RAM, 700G Ultra ATA drive and dual Xeon (3GHz) processors. The MystiQ engine is a middleware system that functions as a preprocessor and uses a complete approach [21, 137]. The materialized view tools are implemented using approximately 5000 lines of OCaml. After importing all probabilistic materialized views, we tuned the database using only the SQL Server Database Engine Tuning Advisor.

Execution Time Reporting Method We reduced query time variance by executing each query seven times, dropping the highest and lowest times and averaging the remaining five times. In all reported numbers, the variance of the five runs was less than 5% of query execution time.

5.4.2 Question 1: Do Representable and Partially Representable views exist?

In Fig. 5.2(a), we show the percentage of views in each workload that is trivially representable because there are no probabilities in the view *(TRIVIAL)*, representable *(REP)*, non-trivially partially representable *(PARTIAL)* or only trivially partially representable *(NOTREP)*. In iLike's workload, 4 of the 5 views (80%) are representable. Further, 98.5% of the over 100k queries that manipulate uncertain data use the representable views. In synthetic data sets, representable views exist as well. In fact, 50% or more of the views in each data set except for AW are representable. Overall, 63% of views are representable. 45% of the representable views are non-trivially representable. Additionally, almost all views we examined have a non-trivial partial representations (over 95%). We conclude that that representable and partially representable views exist and can be used in practice.

5.4.3 Question 2: Do our techniques make query processing more efficient?

The TPC data set is the basis for our performance experiments because it is reproducible and the data can be scaled arbitrarily. We present queries 5 and 10, because they both have many joins (6 and 4) and they are contrasting: Query 10 is *safe* [46, 135], and so can be efficiently evaluated by a modified SQL query. Query 5 is *unsafe* and so requires expensive Monte Carlo techniques. Graphs 5.2(b) and 5.2(c) report the time taken to execute the query and retrieve the results. For query 10, this is the total time for execution because it is safe. In contrast, query 5 requires additional Monte Carlo techniques to compute output probabilities.

Graph Discussion In Fig. 5.2(b), we see running times of query 10 without probabilistic semantics *(PTPC)*, as a safe plan *(SAFE)*, with a subview materialized and retaining lineage *(LIN)* and the same subview without lineage *(NOLIN)*. *LIN* is equivalent to a standard materialized view optimization; the lineage information is computed and stored as a table. In *NOLIN*, we discard the lineage and retain only the probability that a tuple appears in the view. The graph confirms that materializing the lineage yields an order of magnitude improvement for safe queries because we do not need to compute three of the four joins at query execution time. Interestingly, the bars for *NOLIN* show that precomputing the probabilities and ignoring the lineage yields an additional order of magnitude improvement. This optimization is correct because the materialized view is *representable*. This is

interesting because it shows that being aware of when we can remove lineage is helpful even for safe plans.

As a baseline, Fig. 5.2(c) shows the query execution times for query 5 without probabilistic semantics but using the enlarged probabilistic tables *(PTPC)*. Fig. 5.2(c) also shows the cost of retrieving the tuples necessary for Monte Carlo simulation *(MC)*. Similarly, we also see the cost when materializing a view and retaining lineage *(LIN)* and when we precompute the probabilities and discard the lineage *(NOLIN)*. For *(MC)* and *(LIN)*, the extra step of Monte Carlo Simulation is necessary which for TPC 0.1 (resp. TPC 0.5, TPC 1) requires an additional 13.62 seconds (resp. 62.32s, 138.21s). Interestingly, query 5 using the materialized view does *not* require Monte Carlo Simulation because the rewritten query is safe. Thus, the time for *NOLIN* is an end-to-end running time and so we conclude that our techniques offer four order of magnitude improvement over materializing the lineage alone ($8.2s + 138.21s$ with lineage v. 0.03s without).

5.4.4 Question 3: How costly are our algorithms?

All views listed in this paper were correctly classified by our practical algorithm, which always executes in well under 1 second. Finding all representable or partially representable sub-views for all but two queries completed in under 145 seconds; the other two queries completed in under an hour. Materializing views for unsafe queries completed under 1.5 hours for all results reported in the paper. However, this is an offline process and can be parallelized because it can utilize multiple separate Monte Carlo processes.

Chapter 6

VIEW-BASED TECHNIQUE II: APPROXIMATE LINEAGE

In probabilistic databases, *lineage* is fundamental to processing probabilistic queries and understanding the data. Many state-of-the-art systems use a *complete approach*, e.g., Trio [16] or MYSTIQ [46, 137], in which the lineage for a tuple t is a Boolean formula which represents all derivations of t. The central observation in this chapter is that, for many applications, it is often unnecessary for the system to painstakingly track every derivation. A consequence of ignoring some derivations is that our system may return an approximate query probability such as 0.701 ± 0.002, instead of the true value of 0.7. An application may be able to tolerate this difference, especially if the approximate answer can be obtained significantly faster. A second issue is that although a complete lineage approach explains all derivations of a tuple, it does not tell us which facts are the most influential in that derivation. In large data sets, a derivation may become extremely large because it aggregates together a large number of individual facts. This makes determining which individual facts are influential an important and non-trivial task.

With these observations in mind, we advocate an alternative to complete lineage called *approximate lineage*. Informally, the spirit of approximate lineage is to compress the data by tracking only the most influential facts in the derivation. This approach allows us to both efficiently answer queries, since the data is much smaller, and also to directly return the most important derivations. We motivate our study of approximate lineage by discussing two application domains: (1) large scientific databases and (2) similarity data. We show that approximate lineage can compress the data by up to two orders of magnitude, e.g., 100s of MB to 1MB, while providing high-quality explanations.

6.1 Motivating Scenarios

We discuss some applications of approximate lineage.

Process (P)

	Gene Product	Process	λ
(t_1)	AGO2	"Cell Death"	x_1
(t_2)	AGO2	"Embryonic Development"	x_1
(t_3)	AGO2	"Gland development"	x_2
(t_4)	Aac11	"Cell Death"	x_2
(t_5)	Aac11	"Embroynic Development"	x_3

Annotations (Atoms)

Atom	Code	Description	**P**
x_1	TAS	"Dr. X's PubMed PMID:12593804"	$\frac{3}{4}$
x_2	NAS	"Dr. Y's RA Private Communication"	$\frac{1}{4}$
x_3	IEA	"Inferred from Computational Similarity"	$\frac{1}{8}$

$V(y) :- \mathrm{P}(x, y), \mathrm{P}(\text{'Aac11'}, y),\ x \neq \text{'Aac11'}$

V is a view that asks for "*Gene Products that share a process with a product 'Aac11"'*

Level I DB (Complete Lineage)

	Gene Product	λ
(t_6)	AGO2	$(x_1 \wedge x_2) \vee (x_1 \wedge x_3)$

Level II DB (Approximate Lineage)

Type	Lineage Formula
Sufficient	$\tilde{\lambda}^S_{t_6} = x_1 \wedge x_2$
Arithmetization	$\tilde{\lambda}^A_{t_6} = x_1(1-(1-x_2)(1-x_3))$
Polynomial	$\tilde{\lambda}^P_{t_6} = \frac{33}{128} + \frac{21}{32}(x_2 - \frac{1}{4}) + \frac{9}{16}(x_3 - \frac{1}{8})$

Figure 6.1: `Process` (P) relates each gene product to a process, e.g., AG02 is involved in "cell death". Each tuple in `Process` has an annotation from the set of atoms. An atom, x_i for $i = 1, 2, 3$, is a piece of evidence that has an associated probability, e.g., x_1 is the proposition that we trust *"Dr. X.'s PubMed article PMID:12593804"*, which we assign probability $\frac{3}{4}$. *V* is a view that asks for "*Gene Products that share a process with a product 'Aac11"'*. Below *V*'s definition is its output in the original database with a complete approach. At the right examples of approximate lineage functions we consider are listed. The compressed database is obtained by replacing λ with one of these functions, e.g., $\tilde{\lambda}^S_{t_6}$. This database is inspired by the Gene Ontology (GO) database [37].The terms (*Level I*) and (*Level II*) are specific to our approach and defined in (Section 6.1.1).

Application (1): Large Scientific databases In large scientific databases, lineage is used to integrate data from several sources [24]. These sources are combined by both large consortia, e.g., [37], and single research groups. A key challenge faced by scientists is that facts from different sources may not be trusted equally. For example, the Gene Ontology Database (GO) [37] is a large (4GB) freely available database of genes and proteins that is integrated by a consortium of researchers. For scientists, the most important data stored in GO is a set of associations between proteins and their functions. These associations are integrated by GO from many sources, such as PubMed articles [134], raw experimental data, data from SWISS-PROT [20], and automatically inferred matchings. GO tracks the provenance of each association, using what we call *atoms*. An atom is simply a tuple that contains a description of the source of a statement. An example atom is *"Dr. X's PubMed article PMID:12593804"*. Tracking provenance is crucial in GO because much of the data is of relatively low quality: approximately 96% of the more than 19 million atoms stored in GO are automatically inferred. To model these trust issues, our system associates each atom with a probability whose value reflects our trust in that particular annotation. Figure 6.1 illustrates such a database.

Example 6.1.1 A statement derivable from GO is, "Dr. X claimed in PubMed PMID:12593804 that the gene Argonaute2 (AGO2) is involved in cell death" [77]. In our model, one way to view this is that there is a fact, *the gene Argonaute2 is involved in cell death* and there is an atom, *Dr. X made the claim in PubMed PMID:12593804*. If we trust Dr. X, then we assign a high confidence value to this atom. This is reflected in Figure 6.1 since the atom, x_1, has a high probability, $\frac{3}{4}$. More complicated annotations can be *derived*, e.g., via query processing. An example is the view `V` in Figure 6.1, that asks for gene products that share a process with the gene 'Aac11'. The tuple, AGO2 (t_6), that appears in V is derived from the facts that both AGO2 and Aac11 are involved in "cell death" (t_1 and t_4) and "embryonic development" (t_2 and t_5); these tuples use the atoms x_1 (twice), x_2 and x_3 shown in the Annotations table.

A benefit of annotating the data with confidence scores is that the scientist can now obtain the *reliability* of each query answer. To compute the reliability value in a complete approach, we may be forced to process all the lineage for a given tuple. This is challenging, because the lineage can be very large. This problem is not unique to GO. For example, [30] reports that a 250MB biological database has 6GB of lineage. In this thesis, we show how to use *approximate lineage*

to effectively compress the lineage more than two orders of magnitude, even for extremely low error rates. Importantly, our compression techniques allow us to process queries directly on the compressed data. In our experiments, we show that this can result in up to two orders of magnitude more efficient processing than a complete approach.

An additional important activity for scientists is understanding the data; the role of the database in this task is to provide interactive results to hone the scientist's knowledge. As a result, we cannot tolerate long delays. For example, the lineage of even a single tuple in the gene ontology database may be 9MB. Consider a scientist who finds the result of V in Fig 6.1 surprising: One of her goals may be to find out why t_6 is returned by the system, i.e.she wants a sufficient explanation as to why AGO2 was returned. The system would return that the *most likely explanation* is that we trust Dr.X that AGO2 is related to cell death (t_1) and Dr.Y's RA that Aac11 is also related to cell death (t_4). An alternative explanation uses t_1 and the automatic similarity computation (t_5). However, the first explanation is more likely, since the annotation associated with t_4 (x_2) is more likely than the annotation of t_5 (x_3), here $\frac{1}{4} = p(x_2) \geq p(x_3) = \frac{1}{8}$.

A scientist also needs to understand the effect of her trust policy on the reliability score of t_6. Specifically, she needs to know which atom is the most *influential* to computing the reliability for t_6. In this case, the scientist is relatively sure that AGO2 is associated with cell death, since it is assigned a score of $\frac{3}{4}$. However, the key new clement leading to this surprising result is that Aac11 is also associated "cell death", which is supported by the atom x_2, the statement of Dr. Y's RA. Concretely, x_2 is the most influential atom because changing x_2's value will change the reliability of t_6 more than changing any other atom. In our experiments, we show that we can find sufficient explanations with high precision, e.g., we can find the top 10 influential explanations with between 70% and 100% accuracy. Additionally, we can find influential atoms with high precision (80% – 100% of the top 10 influential atoms). In both cases, we can conduct these exploration tasks without directly accessing the raw data.

Application (2): Managing Similarity Scores Applications that manage similarity scores can benefit from approximate lineage. Such applications include managing data from object reconciliation procedures [6, 89] or similarity scores between users, such as iLike.com. In iLike, the system automatically assigns a music compatibility score between friends. The similarity score between

two users, e.g., Bob and Joe, has a lineage: It is a function of many atomic facts, e.g., which songs they listen to and how frequently, which artists they like, etc. All of these atomic facts are combined into a single numeric score which is then converted into *quantitative* buckets, e.g., high, medium and low. Intuitively, to compute such rough buckets, it is unnecessary to precisely maintain every bit of lineage. However, this painstaking computation is required by a complete approach. In this chapter, we show how to use approximate lineage to effectively compress object reconciliation data in the IMDB database [173].

6.1.1 Overview of our Approach

At a high level, both of our example applications, large scientific data and managing similarity scores, manage data that is annotated with probabilities. In both applications, we propose a two-level architecture: The *Level I* database is a large, high-quality database that uses a complete approach and is queried infrequently. The *Level II* database is much smaller, and uses an approximate lineage system. A user conducts her query and exploration tasks on the *Level II* database, which is the focus of this chapter.

The key technical idea of this chapter is *approximate lineage*, which is a strict generalization of complete lineage. Abstractly, lineage is a function λ that maps each tuple t in a database to a Boolean formula λ_t over a fixed set of Boolean atoms. For example in Figure 6.1, the lineage of the tuple t_6 is $\lambda_{t_6} = (x_1 \wedge x_2) \vee (x_1 \wedge x_3)$. In this chapter, we propose two instantiations of approximate lineage: a conservative approximation, *sufficient lineage*, and a more aggressive approximation, *polynomial lineage*.

In *sufficient lineage*, each lineage function is replaced with a smaller formula that logically implies the original. For example, a sufficient lineage for t_6 is $\tilde{\lambda}^S_{t_6} = x_1 \wedge x_2$. The advantage of sufficient lineage is that it can be much smaller than standard lineage, which allows query processing and exploration takes to proceed much more efficiently. For example, in our experiments processing a query on an uncompressed data took 20 hours, while it completed in 30m on a database using sufficient lineage. Additionally, understanding query reliability is easy with sufficient lineage: the reliability computed for a query q is always less than or equal to the reliability computed on the original Level I database. However, only monotone lineage functions can be represented by a sufficient

approach.

The second generalization is *polynomial lineage* which is a function that maps each tuple t in a database to a *real-valued polynomial* on Boolean variables, denoted $\tilde{\lambda}_t^P$. An example polynomial lineage is $\tilde{\lambda}_{t_6}^P$ in Figure 6.1. There are two advantages of using real-valued polynomials instead of Boolean-valued functions: (1) powerful analytic techniques already exist for understanding and approximating real-valued polynomials, e.g., Taylor series or Fourier Series, and (2) *any* lineage function can be represented by polynomial approximate lineage. Polynomial lineage functions can allow a more accurate semantic than sufficient lineage in the same amount of space, i.e., the difference in value between computing q on the Level I and Level II database is small. In Section 6.5 we demonstrate a view in GO such that polynomial lineage achieves a compression ratio of $171:1$ and sufficient lineage achieves $27:1$ compression ratio with error rate less than 10^{-3} (Def. 6.2.10).

Although polynomial lineage can give better compression ratios and can be applied to a broader class of functions, there are three advantages of sufficient lineage over polynomial lineage: (1) sufficient lineage is syntactically identical to complete lineage, and so can be processed by existing probabilistic relational databases without modification, e.g., Trio and Mystiq. (2) The semantic of sufficient lineage is easy to understand since the value of a query is a lower bound of the true value, while a query may have either a higher or lower value using polynomial lineage. (3) Our experiments show that sufficient lineage is less sensitive to skew, and can result in better compression ratios when the probability assignments to atoms are very skewed.

In both lineage systems, there are three fundamental technical challenges: creating it, processing it and understanding it. In this chapter, we study these three fundamental problems for both forms of approximate lineage.

6.1.2 Contributions, Validation and Outline

In the remainder of this chapter, we show that we can (1) efficiently construct both types of approximate lineage, (2) process both types of lineage efficiently and (3) use approximate lineage to explore and understand the data.

- In Section 6.2, we define the semantics of approximate lineage, motivate the technical problems that any approximate lineage system must solve and state our main results. The tech-

nical problems are: creating approximate lineage (Prob. 3); explaining the data, i.e.finding sufficient explanations (Prob. 4), finding influential variables (Prob. 5); and query processing with approximate lineage (Prob. 6).

- In Section 6.3, we define our implementation for one type of approximate lineage, *sufficient lineage*. This requires that we solve the three problems above: we give algorithms to construct it (Section 6.3.2), to use it to understand the data (Section 6.3.3), and to process further queries on the data (Section 6.3.4).
- In Section 6.4, we define our proposal for *polynomial approximate lineage*; our proposal brings together many previous results in the literature to give algorithms to construct it (Section 6.4.2), to understand it (Section 6.4.3) and to process it.
- In Section 6.5, we provide experimental evidence that both approaches work well in practice; in particular, we show that approximate lineage can compress real data by orders of magnitude even with very low error, (Section 6.5.2), provide high quality explanations (Section 6.5.3) and provide large performance improvements (Section 6.5.4). Our experiments use data from the Gene Ontology database [37,156] and a probabilistic database of IMDB [173] linked with reviews from Amazon.

6.2 Statement of Results

We first give some background on lineage and probabilistic databases, and then formally state our problem with examples.

6.2.1 Preliminaries: Queries and Views

In this chapter, we consider conjunctive queries and views written in a datalog-style syntax. A query q is a conjunctive rule written $q :- g_1, \ldots, g_n$ where each g_i is a subgoal, that is, a relational predicate. For example, $q_1 :- R(x), S(x, y, \text{'a'})$ defines a query with a join between R and S, a variable y that is projected away, and a constant 'a'. For a relational database W, we write $W \models q$ to denote that W entails q.

6.2.2 Lineage and Probabilistic Databases

We again adopt a viewpoint of lineage similar to *c*-tables [82, 93], and we think of lineage as a constraint that tells us which worlds are possible. We generalize lineage. To make our generalization clear, we recall the definitions of lineage from Chapter 2.

Definition 6.2.1 (Lineage Function)**.** *An* atom *is a Boolean proposition about the real world, e.g., Bob likes Herbie Hancock. Fix a relational schema σ and a set of atoms $\mathcal{A}$. A lineage function, λ, assigns to each tuple t conforming to some relation in σ, a Boolean expression over $\mathcal{A}$, which is denoted λ_t. An* assignment *is a function $\mathcal{A} \to \{0, 1\}$. Equivalently, it is a subset of $\mathcal{A}$, denoted A, consisting of those atoms that are assigned true.*

Figure 6.1 illustrates tuples and their lineages. The atoms represent propositions about data provenance. For example, the atom x_1 in Figure 6.1 represents the proposition that we trust "Dr. X's PubMed PMID:12593804". Of course, atoms can also represent more coarsely grained propositions, *"A scientist claimed it was true"* or finely-grained facts *"Dr. X claimed it in PubMed 18166081 on page 10"*. In this chapter, we assume that the atoms are given; we briefly discuss this at the end the current section. This form of lineage is sometimes called *Boolean pc-tables*, since we are restricting the domain of every atom to be Boolean.

To define the standard semantics of lineage, we define a *possible world W* through a two-stage process: (1) select a subset of atoms, *A*, i.e., an assignment, and (2) For each tuple *t*, if $\lambda_t(A)$ evaluates to true then *t* is included in *W*. This process results in an unique world *W* for any choice of atoms *A*.

Example 6.2.2 If we select $A_{13} = \{x_1, x_3\}$, that is, we trust Dr. X and Dr. Y's RA, but distrust the similarity computation, then $W_{1256} = \{t_1, t_2, t_5, t_6\}$ is the resulting possible world. The reason is that for each $t_i \in W_{1256}$, λ_{t_i} is satisfied by the assignment corresponding to A_{13} and for each $t_j \notin W_{1256}$, λ_{t_j} is false. In contrast, $W_{125} = \{t_1, t_2, t_5\}$ is *not* a possible world because in W_{125}, we know that AGO2 and Aac11 are both associated with Cell Death, and so AGO2 should appear in the view (t_6). In symbols, $\lambda_{t_6}(W_{125}) = 1$, but $t_6 \notin W_{125}$.

We capture this example in the following definition:

Definition 6.2.3. *Fix a schema σ. A* world *is a subset of tuples conforming to σ. Given a set of atoms A and a world W, we say that W is a* possible world *induced by A if it contains exactly those tuples consistent with the lineage function, that is, for all tuples t, $\lambda_t(A) \iff t \in W$. Moreover, we write $\lambda(A, W)$ to denote the Boolean function that takes value 1 if W is a possible world induced by A. In symbols,*

$$\lambda(A, W) \stackrel{\text{def}}{=} \bigwedge_{t:t\in W} \lambda_t(A) \bigwedge_{t:t\notin W} (1 - \lambda_t(A)) \tag{6.1}$$

Eq. 6.1 is important, because it is the main equation that we generalize to get semantics for approximate lineage.

We complete the construction of a probabilistic database as a distribution over possible worlds. We assume that there is a function p that assigns each atom $a \in \mathcal{A}$ to a probability score denoted $p(a)$. In Figure 6.1, x_1 has been assigned a score $p(x_1) = \frac{3}{4}$, indicating that we are very confident in Dr. X's proclamations. An important special case is when $p(a) = 1$, which indicates absolute certainty.

Definition 6.2.4. *Fix a set of atoms $\mathcal{A}$. A* probabilistic assignment *p is a function from $\mathcal{A}$ to $[0, 1]$ that assigns a probability score to each atom $a \in \mathcal{A}$. A* probabilistic database *$\mathcal{W}$ is a probabilistic assignment p and a lineage function λ that represents a distribution μ over worlds defined as:*

$$\mu(W) \stackrel{\text{def}}{=} \sum_{A\subseteq\mathcal{A}} \lambda(A, W) \left(\prod_{i:a_i\in A} p(a_i) \right) \left(\prod_{j:a_j\notin A} (1 - p(a_j)) \right)$$

Given any Boolean query q on $\mathcal{W}$, the marginal probability of q denoted $\mu(q)$ is defined as

$$\mu(q) \stackrel{\text{def}}{=} \sum_{q:W\models q} \mu(W) \tag{6.2}$$

i.e., the sum of the weights over all worlds that satisfy q.

Since for any A, there is a unique W such that $\lambda(A, W) = 1$, μ is a probability measure. In all of our semantics, the semantic for queries will be defined similarly to Eq. 6.2.

Example 6.2.5 Consider a simple query on our database:

$$q_1() \text{ :- } \mathtt{P}(x, \text{'Gland Development'}), \mathtt{V}(x)$$

This query asks if there exists a gene product x, that is associated with 'Gland Development', and also has a common function with 'Aac11', that is it also appears in the output of V. On the data in Figure 6.1, q_1 is satisfied on a world W if and only if (1) *AGO2 is associated with Gland development* and (2) *AGO2 and Aac11 have a common function*, here, either Embryonic Development or Cell Death. The subgoal requires that t_3 be present and the second that t_6 be present. The formula $\lambda_{t_3} \wedge \lambda_{t_6}$ simplifies to $x_1 \wedge x_2$, i.e., we must trust both Dr.X and Dr.Y's RA to derive q_1, which has probability $\frac{3}{4} \cdot \frac{1}{4} = \frac{3}{16} \approx 0.19$.

We now generalize the standard (complete) semantics to give approximate semantics; the approximate lineage semantics are used to give semantics to the compressed Level II database.

Sufficient Lineage Our first form of approximate lineage is called *sufficient lineage*. The idea is simple: Each λ_t is replaced by a Boolean formula $\tilde{\lambda}_t^S$ such that $\tilde{\lambda}_t^S \implies \lambda_t$ is a tautology. Intuitively, we think of $\tilde{\lambda}_t^S$ as a good approximation to λ_t if λ_t^S and λ_t agree on most assignments. We define the function $\tilde{\lambda}^S(A, W)$ following Eq. 6.1:

$$\tilde{\lambda}^S(A, W) \stackrel{\text{def}}{=} \bigwedge_{t:t\in W} \tilde{\lambda}_t^S(A) \bigwedge_{t:t\notin W} (1 - \tilde{\lambda}_t^S(A)) \tag{6.1s}$$

The formula simply replaces each tuple t's lineage, λ_t with sufficient lineage, $\tilde{\lambda}_t^S$ and then checks whether W is a possible world for A given the sufficient lineage.This, in turn, defines a new probability distribution on worlds $\tilde{\mu}^S$:

$$\tilde{\mu}^S(W) \stackrel{\text{def}}{=} \sum_{A\subseteq\mathcal{A}} \tilde{\lambda}^S(A, W) \left(\prod_{i:a_i\in A} p(a_i) \right) \left(\prod_{j:a_j\notin A} (1 - p(a_j)) \right)$$

Given a query q, we define $\tilde{\mu}^S(q)$ exactly as in Eq. 6.2, with μ syntactically replaced by $\tilde{\mu}^S$, i.e., as a weighted sum over all worlds W satisfying q. Two facts are immediate: (1) $\tilde{\mu}^S$ is a probability measure and (2) for any a conjunctive (monotone) query q, $\tilde{\mu}^S(q) \leq \mu(q)$. Sufficient lineage is syntactically the same as standard lineage. Hence, it can be used to process queries with existing relational probabilistic database systems, such as Mystiq and Trio. If the lineage is a large DNF formula, then any single disjunct is a sufficient lineage. However, there is a trade off between choosing sufficient lineage that is small and lineage that is a good approximation. In some cases,

it is possible to get both. For example, the lineage of a tuple may be less than 1% of the original lineage, but still be a very precise approximation.

Example 6.2.6 We evaluate q from Ex. 6.2.5. In Figure 6.1, a sufficient lineage for tuple t_6 is trusting Dr. X and Dr. Y's RA, that is $\tilde{\lambda}^S_{t_6} = x_1 \wedge x_2$. Thus, q is satisfied exactly with this probability which is ≈ 0.19. Recall from Ex. 6.2.5 that in the original data, q reduced to exactly the formula $x_1 \wedge x_2$, and so the sufficient lineage approach computes the exact answer. In general, this is not the case: if we had chosen $\tilde{\lambda}^S_{t_6} = x_1 \wedge x_3$, i.e., our explanation was trusting Dr.X and the matching, then we would have computed that $\tilde{\mu}^S(q) = \frac{3}{4} \cdot \frac{1}{8} = \frac{3}{32} \approx 0.09 \leq 0.19$.

One can also consider the dual form of sufficient lineage, *necessary lineage*, where each formula λ_t is replaced with a Boolean formula $\tilde{\lambda}^N_t$, such that $\lambda_t \implies \tilde{\lambda}^N_t$ is a tautology. Similar properties hold for necessary lineage: For example, $\tilde{\mu}^N$ is an upper bound for μ, which implies that using necessary and sufficient lineage in concert can provide the user with a more robust understanding of query answers. For the sake of brevity, we shall focus on sufficient lineage for the remainder of this chapter.

Polynomial Lineage In contrast to both standard and sufficient lineages that map each tuple to a Boolean function, polynomial approximate lineage maps each tuple to a *real-valued function*. This generalization allows us to leverage approximation techniques for real-valued functions, such as Taylor and Fourier series.

Given a Boolean formula λ_t on Boolean variables $x_1, \ldots, x_n$ an *arithmetization* is a real-valued polynomial $\lambda^A_t(x_1, \ldots, x_n)$ in real variables such that (1) each variable x_i has degree 1 in λ^A_t and (2) for any $x_1, \ldots, x_n \in \{0, 1\}^n$, we have $\lambda_t(x_1, \ldots, x_n) = \lambda^A_t(x_1, \ldots, x_n)$ [121, p. 177]. For example, an arithmetization of $xy \vee xz$ is $x(1 - (1 - y)(1 - z))$ and an arithmetization of $xy \vee xz \vee yz$ is $xy + xz + yz - 2xyz$. Figure 6.1 illustrates an arithmetization of the lineage formula for t_6, which is denoted $\lambda^A_{t_6}$.

In general, the arithmetization of a lineage formula may be exponentially larger than the original lineage formula. As a result, we do not use the arithmetization directly; instead, we approximate it. For example, an approximate polynomial for λ_{t_6} is $\tilde{\lambda}^P_{t_6}$ in Figure 6.1.

To define our formal semantics, we define $\tilde{\lambda}^P(A, W)$ generalizing Eq. 6.1 by allowing $\tilde{\lambda}^P$ to assign a real-valued, as opposed to Boolean, weight.

$$\tilde{\lambda}^P(A, W) \stackrel{\text{def}}{=} \prod_{t:t\in W} \tilde{\lambda}^P_t(A) \prod_{t:t\notin W} (1 - \tilde{\lambda}^P_t(A)) \tag{6.1p}$$

The first difference in polynomial lineage is that it assigns real-valued weights to worlds, as opposed to Boolean weights. A second difference is that for sufficient and exact lineage approaches once we know the assignment, this determines a unique world W such that $\lambda(A, W)$ is non-zero, i.e., λ is functional in its second argument. In contrast, in polynomial lineage, $\tilde{\lambda}^P$ is a relation: *many worlds* may receive non-zero weight from the same assignment.

Example 6.2.7 In Figure 6.1, W_{1256} is a possible world since $\lambda(A, W_{1256}) = 1$ for $A = \{x_1, x_3\}$. In contrast, $\tilde{\lambda}^P(A, W_{1256}) \neq 1$. To see this, $\tilde{\lambda}^P(A, W_{1256})$ simplifies to $\tilde{\lambda}^P_{t_6}(A, W_{1256})$, since all other lineage functions have $\{0, 1\}$ values. Evaluating $\tilde{\lambda}^P_{t_6}(A)$ gives $\frac{33}{128} + \frac{21}{32}(0 - \frac{1}{4}) + \frac{9}{16}(1 - \frac{1}{8}) = \frac{75}{128} \approx 0.58$. Further, approximate lineage functions may assign non-zero mass even to worlds which are not possible. For example W_{125} is not a possible world, but $\tilde{\lambda}^P(A, W_{13}) = 1 - \lambda_{t_6}(A)(1 - \frac{75}{128}) \neq 0$.

The second step in the standard construction is to define a probability measure μ (Def. 6.2.4); In approximate lineage, we define a function $\tilde{\mu}^P$ – *which may not be a probability measure* – that assigns arbitrary real-valued weights to worlds. Here, $p_i = p(a_i)$ where p is a probability assignment as in Def. 6.2.4:

$$\tilde{\mu}^P(W) \stackrel{\text{def}}{=} \sum_{A\subseteq\mathcal{A}} \tilde{\lambda}(A, W) \left(\prod_{i:a_i\in A} p_i \right) \left(\prod_{j:a_j\notin A} (1 - p_j) \right) \tag{6.3}$$

Our approach is to search for $\tilde{\lambda}^P$ that is a good approximation, that is if for any q, we have $\tilde{\mu}(q) \approx \mu(q)$, i.e., the value computed using approximate lineage is close to the standard approach. Similar to sufficient lineage, we get a query semantic by syntactically replacing μ by $\tilde{\mu}^P$ in Eq. 6.2. However, the semantics for polynomial lineage is more general than the two previous semantics, since an assignment is allowed map to *many* worlds.

Example 6.2.8 Continuing Ex. 6.2.7, in the original data, $\mu(W_{1256}) = \frac{9}{128}$. However, $\tilde{\mu}^P$ assigns

W_{1256} a different weight:

$$\tilde{\mu}^P(W_{1256}) = \tilde{\lambda}^P(W_{1256})\left(\frac{3}{4}\right)\left(\frac{1}{8}\right)\left(1-\frac{1}{4}\right) = \frac{75}{128}$$

Recall q_1 from Ex. 6.2.5; its value is $\mu(q_1) \approx 0.19$. Using Eq. 6.3, we can calculate that the value of q_1 on the Level II database using polynomial lineage in Figure 6.1 is $\tilde{\mu}^P(q_1) \approx 0.17$. In this case the error is ≈ 0.02. If we had treated the tuples in the database independently, we would get the value $\frac{1}{4} * \frac{11}{32} \approx 0.06$ – an error of 0.13, which is an order of magnitude larger error than an approach using polynomial lineage. Further, $\tilde{\lambda}^P$ is smaller than the original Boolean formula.

6.2.3 Problem Statements and Results

In our approach, the original Level I database, that uses a complete lineage system, is lossily compressed to create a Level II database, that uses an approximate lineage system; we then perform all querying and exploration on the Level II database. To realize this goal, we need to solve three technical problems (1) create a "good" Level II database, (2) provide algorithms to explore the data given the approximate lineage, and (3) process queries using the approximate lineage.

Internal Lineage Functions Although our algorithms apply to general lineage functions, many of our theoretical results will consider an important special case of lineage functions called *internal lineage functions* [146]. In internal linage functions, there are some tables (base tables) such that every tuple is annotated with a single atom, e.g., `P` in Figure 6.1. The database also contains derived tables (views), e.g., `V` in Figure 6.1. The lineage for derived tables is derived using the definition of V and tuples in base tables. For our purposes, the significance of internal lineage is that all lineage is a special kind of Boolean formula, a k-monotone DNFs (k-mDNF). A Boolean formula is a k-mDNF if it is a disjunction of monomials each containing at most k literals and no negations. The GO database is captured by an internal lineage function.

Proposition 6.2.9. *If t is a tuple in a view V such that, when unfolded, references k (not necessarily distinct) base tables, then the lineage function λ_t is a k-mDNF.*

Proof. The proof is the same as Proposition 2.3.5 in Chapter 2 with one minor twist: We have

restricted all atoms to occur only positively (since they are Boolean) and so the resulting lineage formula, λ_t is monotone in these atoms. □

One consequence of this is that k is typically small. And so, as in data complexity [1], we consider k a small constant. For example, an algorithm is considered efficient if it is at most polynomial in the size of the data, but possibly exponential in k.

Creating Approximate Lineage

Informally, approximate lineage is good if (1) for each tuple t the function $\tilde{\lambda}_t$ is a close approximation of λ_t, i.e., λ_t and $\tilde{\lambda}_t$ are close on many assignments, and (2) the size of $\tilde{\lambda}_t$ is small for every t. Here, we write $\tilde{\lambda}_t$ (without a superscript) when a statement applies to either type of approximate lineage.

Definition 6.2.10. *Fix a set of atoms $\mathcal{A}$. Given a probabilistic assignment p for $\mathcal{A}$, we say that $\tilde{\lambda}_t$ is an ε-approximation of λ_t if*

$$\mathbf{E}_p[(\tilde{\lambda}_t - \lambda_t)^2] \leq \varepsilon$$

where $\mathbf{E}_p$ denotes the expectation over assignments to atoms induced by the probability function p.

Given a probabilistic assignment p, our goal is to ensure that the lineage function for *every* tuple in the database has an ε-approximation. Def. 6.2.10 is used in computational learning, e.g., [114, 151], because an ε-approximation of a function disagrees on only a few inputs:

Example 6.2.11 Let y_1 and y_2 be atoms such that $p(y_i) = 0.5$ for $i = 1, 2$. Consider the lineage function for some t, $\lambda_t(y_1, y_2) = y_1 \vee y_2$ and an approximate lineage function $\tilde{\lambda}_t^S(y_1, y_2) = \tilde{\lambda}_t^P(y_1, y_2) = y_1$. Here, λ_t and $\tilde{\lambda}_t^S$ (or $\tilde{\lambda}_t^P$) differ on precisely one of the four assignments, i.e., $y_1 = 0$ and $y_2 = 1$. Since all assignments are equally weighted, $\tilde{\lambda}_t^S$ is a $1/4$-approximation for λ. In general, if λ_1 and λ_2 are Boolean functions on atoms $A = \{y_1, \ldots, y_n\}$ such that $p(y_i) = 0.5$ for $i = 1, \ldots, n$, then λ_1 is an ε approximation of λ_2 if λ_1 and λ_2 differ on less than an ε fraction of assignments.

Our first problem is constructing lineage that has arbitrarily small error approximation and occupies a small amount of space.

Problem 3 (Constructing Lineage)**.** *Given a linage function λ_t and an input parameter ε, can we efficiently construct an ε-approximation for λ_t that is small?*

For internal lineage functions, we show how to construct approximate lineage efficiently that is provably small for both sufficient lineage (Section 6.3.2) and polynomial lineage (Section 6.4.2), under the technical assumption that the atoms have probabilities bounded away from 0 and 1, e.g., we do *not* allow probabilities of the form n^{-1} where n is the size of the database. Informally, provably small means that the lineage for a tuple *does not depend on the size of the database*. The lineage of a tuple in the output of a view V may, however, depend (exponentially) on the size of the description of V. Further, we experimentally verify that sufficient lineage offers compression ratios of up to 60 : 1 on real datasets and polynomial lineage offers up to 171 : 1 even with stringent error requirements, e.g., $\varepsilon = 10^{-3}$.

Understanding Lineage

Recall our scientist from the introduction, she is skeptical of an answer the database produces, e.g., t_6 in Figure 6.1, and wants to understand why the system believes that t_6 is an answer to her query. We informally discuss the primitive operations our system provides to help her understand t_6 and then define the corresponding formal problems that we need to solve to apply approximate lineage to this problem.

Sufficient Explanations She may want to know the possible *explanations* for a tuple, i.e., *"why was tuple t_6 returned?"*. Since there are many possible explanations (or derivations), our technical goal is to find the best or *most likely* (top) explanations.

Finding influential atoms Our scientist may want to know which atoms contributed to returning the surprising tuple, t_6. In a complicated query, the query will depend on many atoms, but some atoms are more *influential* in producing the query result than others. Informally, an atom x_1 is influential if it there are many assignments such that it is the "deciding vote", i.e., changing the assignment of x_1 changes whether t_6 is returned. In contrast, an atom that does not effect the answer to the query has no influence. This motivates our technical goal, which is to return the most influential atoms.

The technical challenge in both situation is to perform these actions using approximate lineage on the Level II database, without retrieving the much larger Level I database.

Sufficient Explanations An *explanation* for a lineage function λ_t is a minimal conjunction of atoms τ_t such that for any assignment $\boldsymbol{a}$ to the atoms, we have $\tau_t(\boldsymbol{a}) \implies \lambda_t(\boldsymbol{a})$. The probability of an explanation, τ, is $\mu[\tau]$. A solution would be straightforward on the original, Level I database: Execute the query, produce a lineage formula, and simply select the most highly probable monomials in the answer. Our goal is more difficult: we need to retrieve the *top-k* explanations, ranked by probability, from the lossily-compressed, Level II data.

Problem 4. *Given a tuple t, calculate the top-k explanations, ranked by their probability using only the Level II database.*

This problem is straightforward when using sufficient lineage, but is more challenging for polynomial lineage. The first reason is that polynomials seem to throw away information about monomials. For example, $\tilde{\lambda}^P_{t_6}$ in Figure 6.1 does not mention the terms of *any* monomial. Further complicating matters is that even computing the expectation of $\tilde{\lambda}^P_{t_6}$ may be intractable, and so we have to settle for approximations which introduce error. As a result, we must resort to statistical techniques to guess if a formula τ_t is a sufficient explanation. In spite of these problems, we are able to use polynomial lineage to retrieve sufficient explanations with a precision of up to 70% for $k = 10$ with error in the lineage, $\varepsilon = 10^{-2}$.

Finding Influential Atoms The technical question is: Given a formula, e.g., λ_{t_6}, which atom is most influential in computing λ_{t_6}'s value? We define the *influence* of x_i on λ_t, denoted $\text{Inf}_{x_i}(\lambda_t)$, as:

$$\text{Inf}_{x_i}(\lambda_t) \stackrel{\text{def}}{=} \mu[\lambda_t(A) \neq \lambda_t(A \oplus \{i\})] \tag{6.4}$$

where $\oplus$ denotes the symmetric difference. This definition, or a closely related one, has appeared has appeared in wide variety of work, e.g., underling causality in the AI literature [86, 131], influential variables in the learning literature [114], and critical tuples in the database literature [119, 136].

Example 6.2.12 What influence does x_2 have on tuple t_6 presence, i.e., what is the value of $\text{Inf}_{x_1}(\lambda_{t_6})$? Informally, x_2 can only change the value of λ_{t_6} if x_1 is true and x_3 is false. This happens with prob-

ability $\frac{3}{4}(1-\frac{1}{8}) = \frac{21}{32}$. As we will see later, it is no coincidence this is the coefficient of x_2 in $\tilde{\lambda}^P_{t_6}$: our polynomial representation uses the influence to define its coefficients.

The formal problem is to find the top k most influential variables, i.e., the variables with the k highest influences:

Problem 5. *Given a tuple t, efficiently calculate the k most influential variables in λ_t using only the level II database.*

This problem is challenging because the Level II database is a lossily-compressed version of the database and so some information needed to exactly answer Prob. 5 is not present. The key observation for polynomial lineage is that the coefficients we retain are the coefficients of influential variables; this allows us to compute the influential variables efficiently in many cases. We show that we can achieve an almost-perfect average precision for the top 10. For sufficient lineage, we are able to give an approach with bounded error to recover the influential coefficients.

Query Processing with Approximate Lineage

Our goal is to efficiently answer queries directly on the Level II database, using sampling approaches:

Problem 6. *Given an approximate lineage function $\tilde{\lambda}$ and a query q, efficiently evaluate $\tilde{\mu}(q)$ with low-error.*

Processing sufficient lineage is straightforward using existing complete techniques; However, we are able to prove that the error will be small. We verify experimentally that we can answer queries with low-error 10^{-3}, 2 orders of magnitude more quickly than a complete approach. For polynomial lineage, we are able to directly adapt techniques form the literature, such as Blum *et al.* [19].

6.2.4 Discussion

The acquisition of atoms and trust policies is an interesting future research direction. Since our focus is on large databases, it is impractical to require users to label each atom manual. One approach is

to define a language for specifying trust policies. Such a language could do double duty, by also specifying correlations between atoms. We consider the design of a policy language to be important future work. In this chapter, we assume that the atoms are given, the trust policies are explicitly specified, and all atoms are independent.

6.3 Sufficient Lineage

We define our proposal for sufficient lineage that replaces a complicated lineage formula λ_t, by a simpler (and smaller) formula $\tilde{\lambda}_t^S$. We construct $\tilde{\lambda}_t^S$ using several sufficient explanations for λ_t.

6.3.1 Sufficient Lineage Proposal

Given an internal lineage function for a tuple t, that is, a monotone k-DNF formula λ_t, our goal is to efficiently find a sufficient lineage $\tilde{\lambda}_t^S$ that is small and is an ε-approximation of λ_t (Def. 6.2.10). This differs from *L-minimality* [15] that looks for a formula that is equivalent, but smaller. In contrast, we look for a formula that may only approximate the original formula. More formally, the size of a sufficient lineage $\tilde{\lambda}_t^S$ is the number of monomials it contains, and so is small if it contains few monomials. The definition of ε-approximation (Def. 6.2.10) simplifies for sufficient lineage and gives us intuition how to find good sufficient lineage.

Proposition 6.3.1. *Fix a Boolean formula λ_t and let $\tilde{\lambda}_t^S$ be a sufficient explanation for λ_t, that is, for any assignment A, we have $\tilde{\lambda}_t^S(A) \implies \lambda_t(A)$. In this situation, the error function simplifies to $\mathbf{E}[\lambda_t] - \mathbf{E}[\tilde{\lambda}_t^S]$; formally, the following equation holds $\mathbf{E}[(\lambda_t - \tilde{\lambda}_t^S)^2] = \mathbf{E}[\lambda_t] - \mathbf{E}[\tilde{\lambda}_t^S]$*

Proof. The formula $(\lambda_t - \tilde{\lambda}_t^S)^2$ is non-zero only if $\lambda_t \neq \tilde{\lambda}_t^S$, which means that $\lambda_t = 1$ and $\tilde{\lambda}_t^S = 0$, since $\tilde{\lambda}_t^S(A) \implies \lambda_t(A)$ for any A. Because both λ_t and $\tilde{\lambda}_t^S$ are Boolean, $(\lambda_t - \tilde{\lambda}_t^S)^2 \in \{0, 1\}$ and simplifies to $\lambda_t - \tilde{\lambda}_t^S$. We use linearity of $\mathbf{E}$ to conclude. □

Proposition 6.3.1 tells us that to get sufficient lineage with the low error, it is enough to look for sufficient formula $\tilde{\lambda}_t$ with *high* probability.

Scope of our Analysis In this section, our theoretical analysis considers only internal lineage functions with constant bounded probability distributions; a distribution is *constant bounded* if there

is a constant β such that for any atom a, $p(a) > 0$ implies that $p(a) \geq \beta$. To justify this, recall that in GO, the probabilities are computed based on the type of evidence: For example, a citation in PubMed is assigned 0.9, while an automatically inferred matching is assigned 0.1. Here, $\beta = 0.1$ and is independent of the size of the data. In the following discussion, β will always stand for this bound and k will always refer to the maximum number of literals in any monomial of the lineage formula. Further, we shall only consider sufficient lineage which are subformulae of λ_t. This choice guarantees that the resulting formula is sufficient lineage and is also simple enough for us to analyze theoretically.

6.3.2 Constructing Sufficient Lineage

The main result of this section is an algorithm (Alg. 6.3.2.1) that constructs good sufficient lineage, solving Prob. 3. Given an error term, ε, and a formula λ_t, Alg. 6.3.2.1 efficiently produces an approximate sufficient lineage formula $\tilde{\lambda}_t^S$ with error less than ε. Further, Theorem 6.3.3 shows that the size of the formula produced by Alg. 6.3.2.1 depends only on ε, k and β – not on the number of variables or number of terms in λ_t; implying that the formula is theoretically small.

Before diving into the algorithm, we consider a simple, alternative approach to constructing a sufficient lineage formula to build intuition on the technical.

Example 6.3.2 Given a monotone k-DNF formula λ, suppose that to construct a sufficient lineage formula, we simply select the highest probability monomials. Consider then the following 2-DNF formula, call ϕ_t for $t = 1, \ldots, n, \ldots$.

$$\phi_t \stackrel{\text{def}}{=} \bigwedge_{i=1,\ldots,t} (x_0 \wedge y_i) \ \wedge \ \bigwedge_{j=1,\ldots,t} (x_j \wedge z_j)$$

Let $\varepsilon = 0.1$ (for concreteness), then we set the probabilities as follows: $\mu[x_i] = \mu[z_j] = 0.5$ for $i = 0, \ldots, t$ and $j = 1, \ldots, t$. Then, set $y_i = 0.5 + \varepsilon$ for $i = 1, \ldots, t$.

In ϕ_t the highest probability monomials H_t will be $H = \{(x_0 \wedge y_1), \ldots (x_0 \wedge y_t)\}$. Now, each of these monomials contains the same variable, x_0, and so $\mu[H] \leq Pr[x_0] = 0.5$. On the other hand, the probability of ϕ_t approaches 1, since the other monomials are all independent. Thus, the error of using H approaches 0.5 and in particular cannot be made smaller than a constant. Hence, this

approach would not be a solution for construction for sufficient lineage.

In this example, we should have selected the large independent block – its probability tends to 1. Indeed, this is the general rule: Pick as many independent as you can. What we show next is that when there isn't a sufficiently large independent set then there must be a small set of bottlenecks like x_0, i.e., a small cover, and we can use this cover to recurse.

Algorithm 6.3.2.1 Suff(λ_t, ε) constructs sufficient lineage

Input: A monotone k+1-DNF formula λ_t and an error $\varepsilon > 0$
Output: $\tilde{\lambda}_t^S$, a small sufficient lineage ε-approximation.

1: Find a matching M, greedily. (* A subset of monomials *)
2: **if** $\mu[\lambda_t^S] - \mu[M] \leq \varepsilon$ **then** (* If λ_t is a 1-mDNF always true *)
3: Let $M = m_1 \vee \cdots \vee m_l$ s.t. $i \leq j$ implies $\mu[m_i] \geq \mu[m_j]$
4: **return** $M_r \stackrel{\text{def}}{=} m_1, \ldots, m_r$, r is min s.t. $\mu[\lambda_t] - \mu[M_r] \leq \varepsilon$.
5: **else**
6: Select a small cover $C = \{x_1, \ldots, x_c\} \subseteq$ **var**(M)
7: Arbitrarily assign each monomial to a $x_c \in C$ that covers it
8: **for each** $x_i \in C$ **do**
9: $\lambda_i^S \leftarrow$ **Suff**$(\lambda_i[x_i \rightarrow 1], \varepsilon/c)$. (* $\lambda_i[x_i \rightarrow 1]$ sets $x_1 = 1$ and recurses, and is a k-DNF *)
10: **return** $\bigvee_{i=1,\ldots,n} \lambda_i^S$.

Algorithm Description Alg. 6.3.2.1 is a recursive algorithm, whose input is a k-mDNF λ_t and an error $\varepsilon > 0$, it returns $\tilde{\lambda}_t^S$, a sufficient ε-approximation. For simplicity, we assume that we can compute the expectation of monotone formula exactly. In practice, we estimate this quantity using sampling, e.g., using Luby-Karp [101]. The algorithm has two cases: In case (I) on lines 2-4, there is a large matching, that is, a set of monomials M such that distinct monomials in M do not contain common variables. For example, in the formula $(x_1 \wedge y_1) \vee (x_1 \wedge y_2) \vee (x_2 \wedge y_2)$ a matching is $(x_1 \wedge y_1) \vee (x_2 \vee y_2)$. In Case (II) lines 6-10, there is a small cover, that is a set of variables $C = \{x_1, \ldots, x_c\}$ such that every monomial in λ_t contains some element of C. For example, in $(x_1 \wedge y_1) \vee (x_1 \wedge y_2) \vee (x_1 \wedge y_3)$, the singleton $\{x_1\}$ is a cover. The relationship between the two cases is that if we find a maximal matching smaller than m, then there is a cover of size smaller than km (all variables in M form a cover).

Case I: (lines 2-4) The algorithm greedily selects a maximal matching $M = \{m_1, \ldots, m_l\}$. If M is a

good approximation, i.e., $\mu[\lambda_t^S] - \mu[\bigvee_{m\in N} m] \leq \varepsilon$ then we trim M to be as small as possible so that it is still a good approximation. Observe that $P[\bigvee_{m\in M} m]$ can be computed efficiently since the monomials in M do not share variables, and so are independent. Further, for any size l the subset of M of size l with the highest probability is exactly the l highest monomials.

Case II: (lines 6-10) Let **var**(M) be the set of all variables in the maximal matching we found. Since M is a maximal matching, **var**(M) forms a cover, $x_1, \ldots, x_c$. We then arbitrarily assign each monomial m to one element that covers m. For each x_i, let λ_i be the set of monomials associated to an element of the cover, x_i. The algorithm recursively evaluates on each λ_i, with smaller error, ε/c, and returns their disjunction. We choose ε/c so that our result is an ε approximate lineage.

Theorem 6.3.3 (Solution to Prob. 3)**.** *For any $\varepsilon > 0$, Alg. 6.3.2.1 computes small ε-sufficient lineage. Formally, the output of the algorithm, $\tilde{\lambda}_t$ satisfies two properties: (1) $\tilde{\lambda}_t^S$ is an ε-approximation of λ_t and (2) the number of monomials in $\tilde{\lambda}_t$ is less than $k!\beta^{-\binom{k}{2}} \log^k(\frac{1}{\varepsilon}) + O\left(\log^{k-1}(\frac{1}{\delta})\right)$, which is independent of the size of λ_t.*

Proof. Claim (1) follows from the preceding algorithm description. To prove claim (2), we inspect the algorithm. In Case I, the maximum size of a matching is upper bounded by $\beta^{-k}\log(\frac{1}{\varepsilon})$ since a matching of size m in a k-dnf has probability at least $1 - (1-\beta^k)^m$; if this value is greater than $1-\varepsilon$, we can trim terms; combining this inequality and that $1 - x \leq e^{-x}$ for $x \geq 0$, completes Case I. In Case II, the size of the cover c satisfies $c \leq k\beta^{-k}\log(\frac{1}{\varepsilon})$. If we let $S(k+1,\varepsilon)$ denote the size of our formula at depth $k+1$ with parameter ε, then it satisfies the recurrence $S(k+1,\varepsilon) = (k+1)\beta^{-(k+1)}\log(\frac{1}{\varepsilon}) \cdot S(k, \varepsilon/c)$, which grows no faster than the claimed formula. □

The recurrence of our algorithm is linear, since no monomial is replicated on each recursion and the depth of the recursion at most k, the recurrence has cost $O(k|\lambda_t|)$ steps. But, we must at each stage compute the probability of a DNF formula, a ♯P-hard problem. For this, we use a randomized solution (either Luby-Karp [101] or a Chernoff Bound on naive random sampling). Randomization introduces the possibility of failure, which we cope by in the standard way: a slight increase in running time.

We observe that the running time of the randomized solution is a function of two things (1) the error, which we can take to be a small constant depending on ε, and (2) the probability of failure, δ. We set δ so small that the algorithm succeeds with high probability. Let c be the number of calls to the randomized algorithm (steps) during execution, then the probability that all of the succeed is $(1-\frac{1}{\delta})^s$. Hence, we need to take $\delta \ll s^{-1}$. To see the effect on the running time: Let t be the running time of iteration i and m_i be the number of terms in the approximation on call i, then, $t_i = O(m \log \frac{1}{\delta})$. Now, we observe that $\sum_i m_i = k\,|\lambda_t|$ as above, and so the running time is $\sum_i t_i = k\,|\lambda_t| \log |\lambda_t|$.

Completeness Our goal is to construct lineage that is small as possible; one may wonder if we can efficiently produce substantially smaller lineage with a different, but still efficient, algorithm. We give evidence that no such algorithm exists by showing that the key step in Alg. 6.3.2.1 is intractable (NP-hard) even if we restrict to internal lineage functions with 3 subgoals, that is $k = 3$. This justifies our use of a greedy heuristic above.

Proposition 6.3.4. *Given a k-mDNF formula λ_t, finding a subformula $\tilde{\lambda}_t^S$ with d monomials such that $\tilde{\lambda}_t^S$ has largest probability among all subformula of λ_t is* NP*-Hard, even if $k = 3$.*

Proof. We reduce from the problem of finding an k-uniform k-regular matching in a 3-hypergraph, which is NP-hard, see [88]. Given (V^1, V^2, V^3, E) such that $E \subseteq V^1 \times V^2 \times V^3$, let each $v \in V^i$ have probability $\frac{1}{2}$. We observe that if there is a matching of size d, then there is a function λ_t' with probability $1-(1-\beta^k)^d$, and every other size d formula that is not a matching has strictly smaller probability. Since we can compute the probability of a matching efficiently, this completes the reduction. □

The reduction is from of finding a matching in a k-uniform k-regular hypergraph. The greedy algorithm is essentially an optimal approximation for this hypergraph matching [88]. Since our problem appears to be more difficult, this suggests – but does not prove – that our greedy algorithm may be close to optimal.

6.3.3 Understanding Sufficient Lineage

Both Prob. 4, finding sufficient explanations, and Prob. 5, finding influential variables deal with understanding the lineage functions: Our proposal for sufficient lineage makes Prob. 4 straightfor-

ward: Since λ_t^S is a list of sufficient explanations, we simply return the highest ranked explanations contained in $\tilde{\lambda}_t^S$. As a result, we focus on computing the influence of a variable given only sufficient lineage. The main result is that we can compute influence with only a small error using sufficient lineage. We do not discuss finding the top-k efficiently; for which we can use prior art, e.g., [137]. We restate the definition of influence in a computationally friendly form (Proposition 6.3.5) and then prove bounds on the error of our approach.

Proposition 6.3.5. *Let x_i be an atom with probability $p(x_i)$ and $\sigma_i^2 = p(x_i)(1 - p(x_i))$. If λ_t is a monotone lineage formula:*

$$\text{Inf}_{x_i}(\lambda_t) = \sigma_i^{-2}\mathbf{E}[\lambda_t(x_i - p(x_i))]$$

Proof. We first arithmetize λ_t and then factor the resulting polynomial with respect to x_i, that is $\lambda_t = f_i x_i + f_0$ where neither f_i nor f_0 contain x_i. We then observe that $\text{Inf}_{x_i}(\lambda_t) = \mathbf{E}[\lambda_t(A \cup \{x_i\}) - \lambda_t(A - \{x_i\})]$ for monotone λ_t. Using the factorization above and linearity, we have that $\text{Inf}_{x_i}(\lambda_t) = \mathbf{E}[f_i]$. On the other hand $\mathbf{E}[\lambda_t(x_i - p(x_i))] = \mathbf{E}[f_i x_i(x_i - p(x_i)) + (x_i - p(x_i))f_0]$, since f_i and f_0 do not contain x_i, this reduces to $\mathbf{E}[f_i]\mathbf{E}[x_i(x_i - p(x_i))] + 0$. Observing that $\mathbf{E}[x_i(x_i - p(x_i))] = \sigma_i^2$ proves the claim. □

The use of Proposition 6.3.5 is that to show that we can compute influence from sufficient lineage with small error:

Proposition 6.3.6. *Let $\tilde{\lambda}_t^S$ be a sufficient ε-approximation of λ_t, then for any $x_i \in \mathcal{A}$ s.t. $p(x_i) \in (0, 1)$, we have the following pair of inequalities*

$$\text{Inf}_{x_i}(\tilde{\lambda}_t^S) - \frac{\varepsilon}{\sigma^2}p(x_i) \leq \text{Inf}_{x_i}(\lambda_t) \leq \text{Inf}_{x_i}(\tilde{\lambda}_t^S) + \frac{\varepsilon}{\sigma^2}(1 - p(x_i))$$

Proof. $\mathbf{E}[\lambda_t(x_i - p(x_i))] = \mathbf{E}[(\lambda_t - \tilde{\lambda}_t^S)(x_i - p(x_i)) + \tilde{\lambda}_t^S(x_i - p(x_i))]$. The first term is lower bounded by $-\varepsilon p(x_i)$ and upper bounded by $\varepsilon(1 - p(x_i))$. Multiplying by σ_i^{-2}, completes the bound, since the second term is $\text{Inf}_{x_i}(\tilde{\lambda}_t^S)$. □

This proposition basically says that we can calculate the influence for *uncertain* atoms. With naïve random sampling, we can estimate the influence of sufficient lineage to essentially any desired

precision. The number of relevant variables in sufficient lineage is small, so simply evaluating the influence of each variable and sorting is an efficient solution to solve Prob. 5.

6.3.4 Query Processing

Existing systems such as Mystiq or Trio can directly process sufficient lineage since it is syntactically identical to standard (complete) lineage. However, using sufficient lineage in place of complete lineage introduces errors during query processing. In this section, we show that the error introduced by query processing is at most a constant factor worse than the error in a single sufficient lineage formula.

Processing a query q on a database with lineage boils down to building a lineage expression for q by combining the lineage functions of individual tuples, i.e., *intensional evaluation* [68, 137]. For example, a join producing a tuple t from t_1 and t_2 produces lineage for t, $\lambda_t = \lambda_{t_1} \wedge \lambda_{t_2}$. We first prove that the error in processing a query q is upper bounded by the number of lineage functions combined by q (Proposition 6.3.7). Naïvely applied, this observation would show that the error grows with the size of the data. However, we observe that the lineage function for a conjunctive query depends on at most constantly many variables; from these two observations it follows that the query processing error is only a constant factor worse.

Proposition 6.3.7. *If $\tilde{\lambda}_1^S$ and $\tilde{\lambda}_2^S$ are sufficient ε approximations for $\tilde{\lambda}_1$ and $\tilde{\lambda}_2$ then, both $\tilde{\lambda}_1^S \wedge \tilde{\lambda}_2^S$ and $\tilde{\lambda}_1^S \vee \tilde{\lambda}_2^S$ are 2ε sufficient approximations.*

Proof. Both formulae are clearly sufficient. We write $\left\|\lambda_1\lambda_2 - \tilde{\lambda}_1^S\tilde{\lambda}_2^S\right\| = \left\|\lambda_1(\lambda_2 - \tilde{\lambda}_2^S) + \tilde{\lambda}_2^S(\lambda_1 - \tilde{\lambda}_1^S)\right\|$, each term is less than ε, completing the bound. □

This proposition is essentially an application of a union bound [121]. From this proposition and the fact that a query q that produces n tuples and has k subgoals has kn logical operations, we can conclude that if all lineage functions are ε_S approximations, then $\mu(q) - \tilde{\mu}^S(q) \leq \varepsilon_S kn$. This bound depends on the size of the data. We want to avoid this, because it implies that to answer queries as the data grows, we would need to continually refine the lineage. The following proposition shows that sometimes there is a choice of sufficient lineage that can do much better job; this is essentially the same idea as in Section 6.3.2:

Lemma 6.3.8. *Fix a query q with k subgoals and $\varepsilon > 0$, there exists a database with sufficient approximate lineage function $\tilde{\lambda}$ such that the the lineage for each tuple t, $\tilde{\lambda}_t$ is of constant size and*

$$\mu(q) - \tilde{\mu}^S(q) \leq \varepsilon$$

Proof. As we have observed, we can evaluate a query by producing an internal lineage function. This means that we can apply Theorem 6.3.3 to show that for any $\delta > 0$, there exists a sub-formula $\tilde{\phi}$ of size $f(k, \delta)$ such that $\mu(q) - \mathbf{E}[\tilde{\phi}] \leq \delta$. We must only ensure that the atoms in these monomials are present. □

This shows that sufficient lineage can be effectively utilized for query processing, solving Prob. 6. It is an interesting open question to find such lineage that works for many queries simultaneously.

Example 6.3.9 Our current lineage approach uses only local knowledge, but we illustrate why some global knowledge may be required to construct lineage that is good for even simple queries. Consider a database with n tuples and a single relation $R = \{t_1, \ldots, t_n\}$ and the lineage of tuple $\lambda_{t_t} = x_0 \vee x_i$. A sufficient lineage database could be $\tilde{\lambda}_{t_i} = x_0$ for each i. Notice, that the query $q :- R(x)$ on the sufficient lineage database is then x_0 while the formulas on the level I database is $x_0 \vee x_1 \cdots \vee x_n$. A potentially much larger probability.

6.4 Polynomial Lineage

In this section, we propose an instantiation of polynomial lineage based on sparse low-total-degree polynomial series. We focus on the problems of constructing lineage and understanding lineage, since there are existing approaches, [19], that solve the problem of sampling from lineage, which is sufficient to solve the query evaluation problem (Problem 6).

6.4.1 Sparse Fourier Series

Our goal is to write a Boolean function as a sum of smaller terms; this decomposition is similar to Taylor and Fourier series decompositions in basic calculus. We recall the basics of Fourier Series on the Boolean Hypercube[1].

[1]For more details, see [114, 125]

In our discussion, we fix a set of independent random variables $x_1, \ldots, x_n$, e.g., the atoms, where $p_i = \mathbf{E}[x_i]$ (the expectation) and $\sigma_i^2 = p_i(1 - p_i)$ (the variance). Let $\mathcal{B}$ be the vector space of real-valued Boolean functions on n variables; a vector in this space is a function $\lambda : \{0,1\}^n \to \mathbb{R}$. Rather than the standard basis for $\mathcal{B}$, we define the Fourier basis for functions. To do so we equip $\mathcal{B}$ with an inner product that is defined via expectation, that is, $\langle \lambda_1, \lambda_2 \rangle \stackrel{\text{def}}{=} \mathbf{E}[\lambda_1 \cdot \lambda_2]$. This inner product induces a norm, $\|\lambda_t\|^2 \stackrel{\text{def}}{=} \langle \lambda_t, \lambda_t \rangle$. This norm captures our error function (see Def. 6.2.10) since $\mathbf{E}[(\lambda_t - \tilde{\lambda}_t^P)^2] = \left\|\tilde{\lambda}_t - \tilde{\lambda}_t^P\right\|^2$. We can now define an orthonormal basis for the vector space using the set of *characters*:

Definition 6.4.1. *For each $\boldsymbol{z} \in \{0,1\}^n$, the* character *associated with $\boldsymbol{z}$ is a function from $\{0,1\}^n \to \mathbb{R}$ denoted $\phi_{\boldsymbol{z}}$ and defined as:*

$$\phi_{\boldsymbol{z}} \stackrel{\text{def}}{=} \prod_{i:z_i=1} (x_i - p_i)\sigma_i^{-1}$$

Since the set of all characters is an orthonormal basis, we can write any function in $\mathcal{B}$ as a sum of the characters. The coefficient of a character is given by projection on to that character, as we define below.

Definition 6.4.2. *The* Fourier transform *of a function λ_t is denoted $\mathcal{F}_{\lambda_t}$ and is a function from $\{0,1\}^n \to \mathbb{R}$ defined as:*

$$\mathcal{F}_{\lambda_t}(\boldsymbol{z}) \stackrel{\text{def}}{=} \langle \lambda_t, \phi_{\boldsymbol{z}} \rangle = \mathbf{E}[\lambda_t \phi_{\boldsymbol{z}}]$$

The Fourier series *of f is defined as $\sum_{\boldsymbol{z} \in \{0,1\}^n} \mathcal{F}_{\lambda_t}(\boldsymbol{z})\phi_{\boldsymbol{z}}(A)$.*

The Fourier series captures λ_t, that is, for any assignment A, $f(A) = \sum_{\boldsymbol{z} \in \{0,1\}^n} \mathcal{F}_{\lambda_t}(\boldsymbol{z})\phi_{\boldsymbol{z}}(A)$. An important coefficient is $\mathcal{F}_{\lambda_t}(\mathbf{0})$, which is the probability (expectation) of λ_t. We give an example of to illustrate the computation of Fourier series:

Example 6.4.3 Let $\lambda_t = x_1 \vee \cdots \vee x_n$, that is, the logical or of independent n random variables. The arithmetization for λ_t is $1 - \prod_{i=1,\ldots,n}(1 - x_i)$. Applying Def. 6.4.2, $\mathcal{F}_{\lambda_t}(\mathbf{0}) = \mathbf{E}[\lambda_t] = 1 - \prod_{i=1,\ldots,n}(1 -$

$p(x_i)$) and for $\mathbf{z} \neq \mathbf{0}$:

$$\begin{aligned}\mathcal{F}_{\lambda_t}(\mathbf{z}) = & \quad \mathbf{E}\left[\phi_{\mathbf{z}}\left(1 - \prod_{i=1,\ldots,n}(1 - x_i)\right)\right] \\ = & \quad \mathbf{E}\left[\phi_{\mathbf{z}} - \left(\prod_{i:z_i=1}\phi_{e_i}(1 - x_i)\right)\left(\prod_{j:z_j=0}(1 - x_j)\right)\right] \\ = & \quad \left(\prod_{i:z_i=1}\sigma_i\right)\left(\prod_{j:z_j=0}(1 - p(x_j))\right)\end{aligned}$$

where for $i = 1, \ldots, n$, $\sigma_i^2 = p(x_i)(1 - p(x_i))$ (the variance of x_i).

Our goal is to get a small, but good approximation; we make this goal precise using sparse Fourier series:

Definition 6.4.4. *An* s-sparse series *is a Fourier series with at most* s *non-zero coefficients. We say* λ *has an* (s, ε) *approximation if there exists an* s*-sparse approximation* $\tilde{\lambda}_t^P$ *such that* $\left\|\lambda_t - \tilde{\lambda}_t^P\right\|^2 \leq \varepsilon$*. A* best s*-sparse series for a function* λ *is the* s*-sparse series that minimizes* ε*.*

Our approach for polynomial lineage is to approximate the lineage for a tuple t, λ_t, by a sparse Fourier series $\tilde{\lambda}_t^P$, ideally an (s, ε)-sparse approximation for small s and ε. Additionally, we want $\tilde{\lambda}_t^P$ to have low total degree (constant) so we can describe its coefficients succinctly (in constant space).

Selecting an approximation The standard approach to approximation using series is to keep only the largest coefficients, which is optimal in this case:

Proposition 6.4.5. *For any Boolean function* λ_t *and any* $s > 0$*, a best* s*-spare approximation for* λ_t *is the* s *largest coefficients in absolute value, ties broken arbitrarily.*

Proof. Let g be any t term approximation and S_G be its non-zero coefficients then we can write: $\|f - g\|^2 = \sum_{S_g}(\mathcal{F}_{\lambda_t}(S) - \mathcal{F}_{\lambda_2}(S))^2 + \sum_{\bar{S_g}}\mathcal{F}_{\lambda_t}(S)^2$. Notice that $S \in S_g$ implies that $\mathcal{F}_{\lambda_t}(S) = \mathcal{F}_{\lambda_2}(S)$, else we could get a strictly better approximation – thus, the best approximation consists of a subset of coefficients in the Fourier expansion. If it does not contain the largest in magnitude, we can switch a term from the right to the left sum, and get a strictly better approximation. Thus, all best approximations are of this form. □

6.4.2 Constructing Lineage

We construct polynomial lineage by searching for the largest coefficients using the KM algorithm [107]. The KM algorithm is complete in the sense that if there is an (s, ε) sparse approximation it finds an only slightly worse $(s, \varepsilon + \varepsilon^2/s)$ approximation. The key technical insight, is that k-DNFs do have sparse (and low-total-degree) Fourier series, [114, 151]. This implies we only need to keep around a relatively few coefficients to get a good approximation. More precisely,

Theorem 6.4.6 ([107, 114, 151])**.** *Given a set of atoms $\mathcal{A} = \{x_1, \ldots, x_n\}$ and a probabilistic assignment p, let $\beta = \min_{i=1,\ldots,n}\{p(x_i), 1 - p(x_i)\}$ and λ_t be a (not necessarily monotone) k-DNF function over $\mathcal{A}$, then there exists an (s, ε)-approximation $\tilde{\lambda}_t^P$ where $s \leq k^{O(k\beta^{-1}\log(\frac{1}{\varepsilon}))}$ and the total degree of any term in $\tilde{\lambda}_t^P$ is bounded by $c_0\beta^{-1}k\log(\frac{1}{\varepsilon})$ where c_0 is a constant. Further, we can construct $\tilde{\lambda}_t^P$ in randomized polynomial time.*

The KM algorithm is an elegant recursive search algorithm. However, a key practical detail is at each step it requires that we use a two-level estimator, that is, the algorithm requires that at each step, we estimate a quantity y_1 via sampling; to compute each sample of y_1, we must, in turn, estimate a second quantity y_2 via sampling. This can be very slow in practice. This motivates us to purpose a cheaper heuristic: For each monomial m, we estimate the coefficient corresponding to each subset of variables of m. For example, if $m = x_1 \wedge x_2$, then we estimate $\mathbf{0}$, $\boldsymbol{e}_1$, $\boldsymbol{e}_2$ and $\boldsymbol{e}_{12}$. This heuristic takes time $2^k|\lambda_t|$, but can be orders of magnitude more efficient in practice, as we show in our evaluation section (Section 6.5.2). This is linear with respect to data complexity.

6.4.3 Understanding Approximate Lineage

Our goal in this section is to find sufficient explanations and influential variables, solving Problem 4 and Problem 5, respectively.

Sufficient Explanations Let λ_t be a lineage formula such that $\mathbf{E}[\lambda_t] \in (0, 1)$ and $\tilde{\lambda}_t^P$ be a polynomial approximation of λ_t. Given a monomial m, our goal is to test if m is a sufficient explanation for λ_t. The key idea is that m is a sufficient explanation if and only if $\mu[\lambda_t \wedge m] = \mu[m]$, since this implies the implication holds for every assignment.

If $\tilde{\lambda}_t^P$ is exactly the Fourier series for λ_t, then we can compute each value in time $O(2^k)$, since

$$\mathbf{E}[\tilde{\lambda}_t^P m] = \sum_{\mathbf{Z}: z_i=1 \implies i \in m} \mathcal{F}_{\lambda_t}(\mathbf{z}) \left(\prod_{i \in m: \mathbf{Z}_i=1} \sigma \right) \left(\prod_{j \in m: \mathbf{Z}_j=0} \mu_j \right) \tag{6.5}$$

However, often λ_t is complicated, which forces us to use sampling to approximate the coefficients of $\tilde{\lambda}_t^P$. Sampling introduces noise in the coefficients. To tolerate noise, we relax our test:

Definition 6.4.7. *Let $\tau > 0$, the tolerance, and $\delta > 0$, the confidence, then we say that a monomial m is a* (τ, δ) sufficient explanation *for $\tilde{\lambda}_t^P$ if:*

$$\mu_{\mathcal{N}}[|\underbrace{\mathbf{E}[\tilde{\lambda}_t^P \cdot m] - \mathbf{E}[m]}_{(\dagger)}| \leq \tau] \geq 1 - \delta \tag{6.6}$$

where $\mathcal{N}$ denotes the distribution of the sampling noise.

The intuition is that we want that $\mathbf{E}[\tilde{\lambda}_t^P m]$ and $\mathbf{E}[m]$ to be close with high probability. For independent random sampling, the $\mathcal{N}$ is a set of normally distributed random variables, one for each coefficient. Substituting Eq. 6.5 into Eq. 6.6 shows that (†) is a sum of 2^k normal variables, which is again normal; we use this fact to estimate the probability that (†) is less than τ.

Our heuristic is straightforward, given a tolerance τ and a confidence δ: For each monomial m, compute the probability in Eq. 6.6, if it is within δ then declare m a sufficient explanation. Finally, rank each sufficient explanation by the probability of that monomial.

Influential tuples The key observation is that the influence of x_i is determined by its coefficient in the expansion [114, 151]:

Proposition 6.4.8. *Let λ_t be an internal lineage function, x_i an atom and $\sigma_i^2 = p(x_i)(1 - p(x_i))$ then*

$$\mathrm{Inf}_{x_i}(\lambda_t) = \sigma_i^{-1} \mathcal{F}_{\lambda_t}(e_i)$$

This gives us a simple algorithm for finding influential tuples using polynomial lineage, simply scale each $\mathcal{F}_{\lambda_t}(e_i)$, sort them and return them. Further, the term corresponding to e_i in the transform is $\mathcal{F}_{\lambda_t}(e_i)\phi_{e_i} = \mathrm{Inf}_{x_i}(\lambda_t)(x_i - p(x_i))$, as was shown in Figure 6.1.

Query	Tables	# Evidence	# Tuples	Avg. Lin. Size	Size
V1	8	2	1	234	12k
V2	6	2	1119	1211	141M
V3	6	1	295K	3.36	104M
V4	7	1	28M	7.68	31G

Figure 6.2: Query statistics for the GO DB [37].

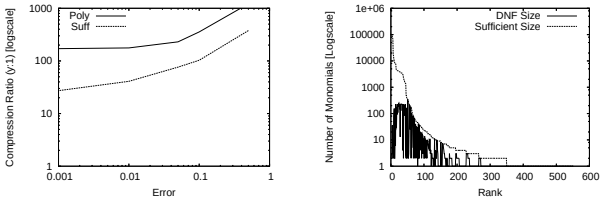
6.5 Experiments

In this section, we answer three main questions about our approach: (1) In Section 6.5.2, do our lineage approaches compress the data? (2) In Section 6.5.3, to what extent can we recover explanations from the compressed data? (3) In Section 6.5.4, does the compressed data provide a performance improvement while returning high quality answers? To answer these questions, we experimented with the Gene Ontology database [37] (GO) and similarity scores from a movie matching database [137, 173].

6.5.1 Experimental Details

Primary Dataset The primary dataset is GO, that we described in the introduction. We assigned probability scores to evidence tuples based on the type of evidence. For example, we assigned a high reliability score (0.9) to a statement in a PubMed article, while we assigned a low score (0.1) to an automated similarity match. Although many atoms are assigned the same score, they are treated as independent events. Additionally, to test the performance of our algorithms, we generated several probability values that were obtained from more highly skewed distributions, that are discussed in the relevant sections.

Primary Views We present four views which are taken from the examples and view definitions that accompany the GO database [37]. The first view `V1` asks for all evidence associated with a fixed pair of gene products. `V2` looks for all terms associated with a fixed gene product. `V3` is a view of all annotations associated with the Drosophila fly (via FlyBase [66]). `V4` is a large view of all gene products and associated terms. Figure 6.2 summarizes the relevant parameters for each view: (1) the number of tables in the view definition (2) the number of sources evidence, that is, how many



(a) (b)

Figure 6.3: (a) Compression ratio as error increases in log scale for query `V2`. (b) Distribution of size of DNF for `V2` with and without compression, x-axis is sorted by size, e.g., $x = 1$ is the largest DNF (823k).

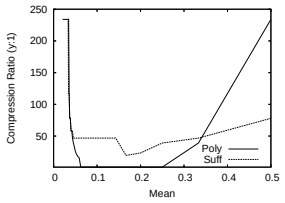
times it joins with the evidence table (3) the number of tuples returned (4) the average of the lineage sizes for each tuple, and (5) the storage size of the result.

Secondary Dataset To verify that our results apply more generally than the GO database, we examined a database that (fuzzily) integrated movie reviews from Amazon [174] that have been integrated with IMDB (the Internet Movie Database) [173]. This data has two sources of imprecision: matches of titles between IMDB and Amazon, ratings assigned to each movie by automatic sentiment analysis, that is, a classifier.

Experimental Setup All experiments were run on a Fedora core Linux machine (2.6.23-14 SMP) with Dual Quad Core 2.66GHz 16Gb of RAM. Our prototype implementation of the compression algorithms was written in approximately 2000 lines of Caml. Query performance was done using a modified C++/caml version of the MystiQ engine [21] backed by databases running SQL Server 2005. The implementation was not heavily optimized.

6.5.2 Compression

We verify that our compression algorithms produce small approximate lineage, even for stringent error requirements. We measured the compression ratios and compression times achieved by our approaches for both datasets at varying errors.



(a)

	Suff	Poly
Q	PT	PT
V1	0.23s	0.50s
V2	3.5h	3.5h
V3	10.3m	24.3m
V4	50.3h	67.4h

(b)

Figure 6.4: (a) The compression ratio versus the mean of the distribution for V1. Sufficient is more stable, though the polynomial lineage can provide better approximation ratios. (b) The compression time for each view, the processing time (PT).

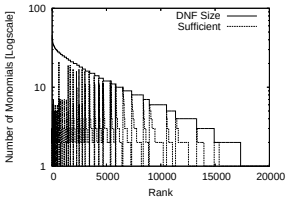
Compression Ratios Figure 6.3(a) shows the compression ratio versus error trade-off achieved by polynomial and sufficient lineage for V2. Specifically, for a fixed error on the x-axis the y axis shows the compression ratio of the lineage (in log scale). As the graph illustrates, in the best case, V2, the compression ratio for the polynomial lineage is very large. Specifically,even for extremely small error rates, 10^{-3}, the compressed ratio $171:1$ for polynomial lineage versus $27:1$ times smaller for sufficient lineage. In contrast, V3 is our worst case. The absolute maximum our methods can achieve is a ratio of $3.36:1$, which is the ratio we would get by keeping a single monomial for each tuple. At an error $\varepsilon = 0.01$, polynomial lineage achieves a $1.8:1$ ratio, while sufficient lineage betters this with a $2.1:1$ ratio.

The abundance of large lineage formula in V2 contain redundant information, which allows our algorithms to compress them efficiently. Figure 6.3(b) shows the distribution of the size of the original lineage formulae and below it the size after compression. There are some very large sources in the real data; the largest one contains approximately 823k monomials. Since large DNFs have probabilities very close to one, polynomial lineage can achieve an ε-approximation can use the constant 1. In contrast, sufficient lineage cannot do this.

Effect of Skew We investigate the effect of skew, by altering the probabilistic assignment, that is, the probability we assigned to each atom. Specifically, we assigned an atom a score drawn from a skewed probability distribution. We then compressed V1 with the skewed probabilities. V1 contains only a single tuple with moderate sized lineage (234 monomials). Figure 6.4(a) shows the

ε	Suff	Poly
0.1	4.0	4.5
0.05	2.7	2.6
0.01	1.4	1.5
0.001	1.07	1.3

(a)



(b)

Figure 6.5: (a) Compression Ratio ($\frac{|\text{Original}|}{|\text{Compressed}|}$) (b) The distribution of lineage size in IMDB view, by rank.

compression ratio as we vary the skew from small means, 0.02, to larger means, 0.5. More formally, the probability we assign to an atom is drawn from a Beta distribution with $\beta = 1$ and α taking the value on the x axis. Sufficient lineage provides lower compression ratios for extreme means, that is close to 0.02 and 0.5, but is more consistent in the less extreme cases.

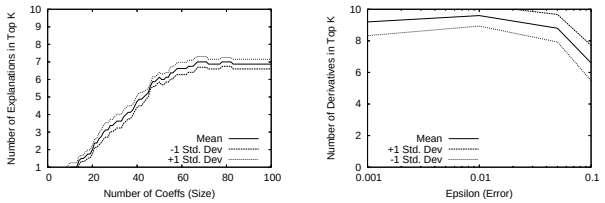
Compression Time Figure 6.4(b) shows the processing time for each view we consider. For views `V2`, `V3` and `V4`, we used 4 dual-core CPUs and 8 processes simultaneously. The actual end-to-end running times are about a factor of 8 faster, e.g., `V2` took less than 30m to compress. It is interesting to to note that the processor time for `V2` is much larger than the comparably sized `V3`, the reason is that the complexity of our algorithm grows non-linearly with the largest DNF size. Specifically, the increase is due to the cost of sampling.

The compression times for polynomial lineage and sufficient lineage are close; this is only true because we are using the heuristic of Section 6.4.2. The generic algorithm is orders of magnitude slower: It could not compress `V1` in an hour, compared to only 0.5s using the heuristic approach. Our implementation of the generic search algorithm could be improved, but it would require orders of magnitude improvement to compete with the efficiency the simple heuristic.

IMDB and Amazon dataset Using the IMDB movie data, we compressed a view of highly rated movies. Figure 6.5(a) shows the compression ratio for versus error rate. Even for stringent error requirements, our approach is able to obtain good compression ratios for both instantiations of approximate lineage. Figure 6.5(b) shows the distribution of the lineage size, sorted by rank, and its

sufficient compression size. Compared to Figure 6.3, there are relatively few large lineage formulae, which means there is less much opportunity for compression. On a single CPU, the time taken to compress the data was always between 180 and 210s. This confirms that our results our more general than a single dataset.

6.5.3 Explanations



(a) (b)

Figure 6.6: (a) Shows the precision of the top-*k* explanations versus the number of terms in the polynomial expansion (c) The number (precision) of influential variables in the top 10 returned using sufficient lineage that are in the top 10 of the uncompressed function.

We assess how well approximate lineage can solve the explanation tasks in practice, that is finding sufficient explanations (Prob. 4) and finding influential variables (Prob. 5). Specifically, we answer two questions: (1) How well can sufficient lineage compute influential variables? (2) How well can polynomial lineage generate sufficient explanations?

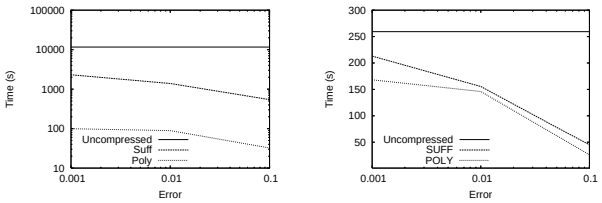
To answer question (1), we created 10 randomly generated probabilistic assignment for the atoms in `V1`; we ensured that the resulting lineage formula had non-trivial reliability, i.e., in $(0.1, 0.9)$. We then tested precision: Out of the top 10 influential variables, how many were returned in the top 10 using sufficient lineage (Section 6.3.3)? Figure 6.6(b) shows that for high error rates, $\varepsilon = 0.1$, we still are able to recover 6 of the top 10 influential variables and for lower error rates, $\varepsilon = 0.01$, we do even better: the average number of recovered top 10 values is 9.6. The precision trails-off for very small error rates due to small swaps in rankings near the bottom of the top 10, e.g., all top 5 are within the top 10.

To answer question (2), we used the same randomly generated probabilistic assignments for the atoms in `V1` as in the answer to question (1). Figure 6.6(a) shows the average number of terms in

the top k explanations returned by the method of Section 6.4.3 that are actual sufficient explanations versus the number of terms retained by the formula. We have an average recall of approximately 0.7 (with low standard deviation), while keeping only a few coefficients. Here, we are using the heuristic construction of polynomial lineage. Thus, this experiment should be viewed as a lower bound on the quality of using polynomial lineage for providing explanations.

These two experiments confirm that both sufficient and polynomial lineage are able to provide high quality explanations of the data directly on the compressed data.

6.5.4 Query Performance



(a) (b)

Figure 6.7: Query performance on (a) `V2`. (b) IMDB data.

Figure 6.7 shows the effect of compression on execution time of `V2`; The query asks to compute each tuple in the view. The y-axis is in log scale, it takes just under 20 hours to run this query on the uncompressed data. On data compressed with sufficient lineage at $\varepsilon = 0.001$, we get an order of magnitude improvement; the query takes approximately 35m to execute. Using the data compressed with polynomial lineage, we get an additional order of magnitude; the query now runs in 1.5m.

Figure 6.7(b) shows the effect of compression on query performance for the IMDB movie dataset where the compression was not as dramatic. Again our query was to compute the lineage for each tuple in the view. The time taking is to perform Monte Carlo sampling on the now much smaller query. As expected, the data with higher error, and so smaller, allows up to a five time performance gain. In this example both running times scale approximately with the size of compression.

Chapter 7

RELATED WORK

We begin by discussing work that deals with managing imprecision in a broad sense, and then we discuss work whose technical details or approach are related.

7.1 Broadly Related Work

We begin with the most closely related work.

Probabilistic Relational Databases

In recent years, the database community has seen a flurry of work probabilistic databases. Managing probabilistic data is difficult, and each of the systems similar problems in different ways. The MystiQ project at the University of Washington was started with the paper by Dalvi and Suciu [46] who showed that a dichotomy holds for conjunctive queries without self-joins: Either the query's data complexity is PTIMEor it is ♯P-hard. Moreover, if the query is in PTIMEthen we can create an SQL query which simply multiplies or adds probabilities to compute the probability efficiently and correctly. Inspired by this result, in this dissertation, we generalize this dichotomy result to aggregate queries in Chapter 4. Later, this result was generalized to queries that contained self-joins; the algorithm was more complicated and the translation to standard SQL queries is currently unknown [47].

At about the same time, the Trio project began to focus on representation issues for uncertain databases, calling itself an "Uncertain and Lineage Database" (ULDB) [16, 17, 145, 146, 168]. This project promoted lineage as not only a technical convenience, but an important concept in its own right [168]. More recently, Das Sarma [146] optimize queries by exploiting common structure in the lineage of queries at run-time. A similar approach was taken in the context of graphical models by Sen *et al.* [149]. In contrast, the techniques in this paper are usually done at *compile-time*, similar to a standard query optimizer. Sen's earlier work [148] in this area was one of the first to suggest

that query evaluation could be done efficiently by casting the problem in term of graphical models. One observation made in this work is that sometimes the data may contain structure unknown to the optimizer (e.g., a functional dependency), which means that a query which has ♯P-data complexity in general, is actually tractable. His approach discovers such structure at runtime. The MayBMS system [8, 9, 91] is also a probabilistic relational database with a representation that is close to the Mystiq system; they do, however, allow a much richer language than discussed in this dissertation. In particular, their query language allows direct predication on probabilities, e.g., *"is the probability of* $q_1 < 0.3$ *and* $q_2 > 0.4$*?"*, and is able to introduce uncertainty through the `repair-by-key` construct. These enhancements are not superficial: the query language they propose captures exactly second-order logic; interestingly, they recently announced an algebra to compute it similar to the relational algebra [106]. Other powerful features of their language allows include expressing conditionals, i.e., *"what is the probability of q given that price > 10?"*. Additionally, the have focused on secondary storage issues and begun an interesting research program of *physical optimization for probabilistic databases* in the context of the SPROUT project [127]. One interesting observation of this project is that the existence of a safe plan for a query can be exploited – even the plan itself is not sued. In this context, they also looked at computing probabilities without exact procedures like Davis-Putnam and Ordered Binary Decision diagrams.

Although there is a lot of recent work on probabilistic relational databases, the problems of imprecision and uncertainty in data are not new: they have a long history within the database community [13, 28, 57, 144] . For example Barbará *et al.* [13] discussed a probabilistic model that is very close to the BID model we discuss here. To keep the evaluation tractable, they did not allow duplicate elimination in their queries; the techniques in this dissertation attempt to continue this line of work by efficiently evaluating a larger set of SQL queries, notably aggregates. Predating this work by more than a decade was Cavallo and Piterali [28] who studied a probabilistic database that captured a single alternative world. In the ProbView system [144], an alternate approach t was taken which allowed interval probabilities to be returned. This had the benefit that a much richer class of queries and aggregates can be handled. In some cases, the intervals may degenerate to $[0, 1]$, which gives little information to the user.

At the same time, there were approaches to study a problem called *query reliability* which implicitly used a probabilistic database [78]. The idea was to understand to what extent a queries

depends on the answers in the database, formally it was a tuple independent database where every tuple was given a probability of $\frac{1}{2}$. The reliability is then the probability that the (Boolean) query is true.

Inconsistent and Incomplete Databases

An alternative approach to probabilistic databases from which this thesis drew inspiration is the area of incomplete and inconsistent databases. For example, we may have two databases who individually are consistent, but when we merge them some dependencies fail, such as a key constraint. In one database, John lives in Seattle, and in the other he lives in New York. Although each is individually consistent, when merged together they become inconsistent. In this context, there has been a great deal of work [18, 69, 79, 93, 165]. The typical query semantic of this work is greatest lower bound or least upper bound on the set of all minimal repairs. Also known as the *certain* or *possible* answer semantics. One particular relevant piece of related work is Arenas *et al.* [11] who consider the complexity of aggregate queries, similar to `HAVING` queries, over data which violates functional dependencies. They also consider multiple predicates, which we do not. There is a deep relationship between the repair semantics and probabilistic approaches. A representative work in this direction is Andristos *et al.* [7]. No discussion of incomplete databases would be complete without the seminal work of Imielinski and Lipski [93] who formalized the notion of *c*-tables and initiated the study of incomplete and inconsistent databases.

Lineage and Provenance

Lineage is central to the Trio system [168], who identifies lineage as a central concept [129]. Our thinking of lineage was influenced by a paper of Green and Tannen [82], which spelled out the connections of the various models in the literature – notably to the *c*-tables of Imielinski and Lipski [93] – to the model we presented here. In addition, view of lineage in this paper was heavily influenced by a paper of Green *et al.* [81] who explained that semirings were the right mathematical structure needed to unify the various concepts of lineage. In this dissertation we view lineage as a formal language to precisely explain our algorithms. In recent years, however, there has been work on the practical utility of provenance and lineage for applications, especially, in scientific

databases [25, 30, 53, 80]. There has been work in the systems community on entire architectures that are provenance aware, e.g., provenance-aware storage systems [122]. This speaks to the fundamental nature of the concept of provenance or lineage.

Succinct models

While the BID formalism is complete, Markov Networks [45] and Bayesian Networks [130], and their extensions to Probabilistic Relational Models [67], allow some discrete distributions to be expressed much more succinctly. The trade-off is that query answering becomes correspondingly more difficult. For example, Roth suggests that inference in even very simply Bayes nets do not admit efficient approximation algorithms. There is active work in the database community about adopting these more succinct models, which represent an interesting alternative approach to the techniques of this dissertation [148, 167]. One clear advantage of this approach is that we it can leverage the very interesting and deep work that has gone on in the graphical model community, such as variational methods [98], differential inference [50], and lifted inference [52, 153]. On the other hand, we contend that by choosing a more restrictive model, we can more fully concentrate on scale. In particular, we are unaware of any of these approaches which runs with comparably performance to a standard relational database, as we have demonstrated in this thesis in Chapter 5. That said, there is a huge opportunity to merge the two approaches in the next few years.

Continuous attributes

In this work, we do not consider continuous attribute values, e.g., the techniques in this dissertation cannot handle the case where the attribute `temperature` has a normal distribution with mean 40. Representing this kind of uncertainty is central to approaches that integrate sensor data such as the BBQ project [55], or the Orion System [34]. More recently the MCDB project has advocated an approaching based on Monte-Carlo sampling which can generate much more sophisticated distributions [95]. This supports, for example, the ability to "what-if" answer queries such as *"if we had lowered our prices by 5%, would would our profit have been?"*; internally, the user has specified a demand curve that relates price changes to expected demand, and the system samples from the resulting distribution many times. This expressive power comes at a price, and the cen-

tral challenge here is performance. The performance challenge is more difficult than in our setting, because the sampling procedure is a black-box; nonetheless, by bundling samples together and late-materialization strategies the MCDB system can achieve good performance. Deshpande *et al.* [56] in the BBQ project consider probabilistic databases resulting from sensor networks so that the database models continuous values, such as temperature. The focus in this work is on rich correlation models, but simpler querying.

Semistructured Models

There is also a wealth of work in non-relational systems, notably in semi-structured XML-based systems [3, 36, 92, 104, 124, 150]. As with relational probabilistic databases, there have been a number of different variants of probabilistic XML proposed in the literature. We refer the reader to Kimelfeld [103, Chapter 4] for a comprehensive taxonomy and comparison of expressive power of these models. One compelling reason to consider probabilistic XML is that there may be ambiguity in the structure of the data, and as noted by Nierman *et al.* [124], XML gracefully captures incompleteness. This property makes XML a particularly appropriate model for data integration applications [163]. Hung *et al.* [92] defines a formal, probabilistic XML algebra that is similar in spirit to the intensional algebra of Fuhr for relational databases [68].

Sequential Models

Another line of work in the database area deals with sequential, relational models called *Markovian Streams or Sequences* [99, 138]. These streams arise from tasks including RFID networks [138], Radar monitoring systems [157], and speech-recognition systems [111]. The work of Kanagal and Deshpande [99] maps a SQL query to operations on a graphical model representing the input, which allows them to leverage the extensive work in the graphical model community [98]. In the the SASE project *et al.* [157] and Lahar projects [138], a regular-expression-like language is used. These projects both build on earlier work in the event processing literature such as Caygua [22] and SnoopIB [4]. The processing techniques are automaton-based, which allows near-real-time performance in some situations. More recently, there has been work on creating indexes for these models such as the Markov Chain Index [112] and its generalization to tree-structured models [100].

The key idea in both approaches is to save or cache some portion of the probabilistic inference that recurs.

Applications

There has been a wealth of interest in probabilistic models in the database community. The Conquer system [7] allowed users to cope with uncertainty arising from entity resolution. Their focus was on efficient evaluation on probabilistic data, which is a common goal of the MYSTIQ project. Gupta and Sarawagi [84] showed that they could model the output of information extraction tasks using the BID model. A key argument they made for using a probabilistic database to manage these tasks is that throwing away low-scoring extractions negatively impacted recall. One major motivation for probabilistic databases is to increase the recall, without losing too much precision. More sophisticated probabilistic models for information extraction are an area of interesting ongoing work, notably the Avatar group [96, 116]. This project is building rich probabilistic models to increase the recall of hand-written extractors. Another important application for probabilistic relational databases is managing the output of entity-resolution or deduplication tasks [6, 10, 32, 65, 71, 83, 89, 169, 170], as we discussed in Chapter 3.

7.2 Specific Related Work

In this section, we discuss work that is very closely to specific technical contributions of this dissertation.

Top-k and Ranking

Soliman *et al.* [154] consider combining top-*k* with measures, such as `SUM`, for example *" Tell me the ten highest ranked products by total sales?"*, when the underlying data is imprecise. This combines both the uncertainty of the probabilities along with the measure dimension and has surprisingly subtle semantics that have been the subject of interesting debate within the community [38, 172]. This work is similar in spirit to our `HAVING` (Chapter 4) and top-*k* processing based on probabilities (Chapter 3). In the original paper of Soliman *et al.*, they considered a rich correlation model (essentially arbitrary Bayes networks), but they do not focus on complex queries involving joins.

This work has inspired a large amount of follow-up work including more efficient algorithms for restricted models [73, 90, 171]. In addition, combining probabilistic data and skyline operators is considered by Pei [132]. There is very interesting recent work that combines ranking (and clustering) in one unified framework with an approach based on generating functions [113].

Materialized Views

Materialized views are a fundamental technique used to optimize queries [2, 33, 75, 85] and as a means to share, protect and integrate data [117, 160] that are currently implemented by all major database vendors. Because the complexity of deciding when a query can use a view is high, there has been a considerable amount of work on making query answering using views algorithms scalable [75, 133]. In the same spirit, we provide efficient practical algorithms for our representability problems. As there are technical connections between our problem and the view-security problem, an interesting problem is to apply the work by Machanavajjhala [115] on expanding the syntactic boundary of tractability to the view setting. In prior art [51], the following question is studied: Given a class of queries $\mathcal{Q}$ is a particular representation formalism closed for all $Q \in \mathcal{Q}$? In contrast, our test is more fine-grained: For any fixed conjunctive Q, is the *BID* formalism closed under Q? A related line of work on World Set Decompositions [9] which allow complete representations by factoring databases; applying the techniques to this representation system is an interesting problem.

Aggregation

Aggregation is a fundamental operation in databases, and it should come as no surprise that aggregation has been considered for probabilistic data many times. In the OLAP setting, Burdick *et al.* [26, 27] give efficient algorithms for *value aggregation* in a model that is equivalent to the single table model. Their focus is on the semantics of the problem. As such, they consider how to assign the correct probabilities, called *the allocation problem*, and handling constraints in the data. The allocation problem is an interesting and important problem. Ross *et al.* [144] describe an approach to computing aggregates on a probabilistic database, by computing bounding intervals (e.g., the `AVG` is between [5600, 5700]). They consider a richer class of aggregation functions than we discuss, but with an incomparable semantics. Their complexity results show that computing bounding intervals

exactly is NP-Hard. In contrast, we are interested in a more fine-grained static analysis: our goal is to find the syntactic boundary of hardness. Trio also uses a bounded interval style approach [123].

There is work on value aggregation on a streaming probabilistic databases [97]. In addition, they consider computing value approximations aggregates, such as AVG, in a streaming manner. In contrast, computing the AVG for predicate aggregates (as we do in Chapter 4) on a single table is ♯P-Hard. One way to put these results together is that computing a value aggregate is the first moment (expectation) while a HAVING aggregate allows us to capture the complete distribution (in the exact case). Kanagal and Deshpande [99] also work in the streaming context of aggregation that computes an expected value style of aggregation. This work does not look at complex queries, like joins. Koch [105] formalizes a language that allows predication on probabilities and discusses approximation algorithms for this richer language, though he does not consider HAVING aggregation. This is in part due to the fact that his aim is to create a fully compositional language for probabilistic databases [106]. Extending our style of aggregation to a fully compositional language is an interesting open problem. The problem of generating a random world that satisfies a constraint is fundamental and is considered by Cohen *et al.* [35]. They point out that many applications for this task, and use it to answer rich queries on probabilistic XML databases. In this paper, we differ in the constraint language we choose and that we use our sampling algorithm as a basis for an FPTRAS.

Our trichotomy results are based on the conjecture that ♯BIS does not have an FPTRAS. Evidence of this conjecture is given by Dyer [58, 59] by establishing that this problem is complete for a class of problems with respect to *approximation preserving reductions*. At this point, it would be fair to say that this conjecture is less well established than ♯P ≠ P. Any positive progress, i.e., showing that ♯BIS does have an FPTRAS, could be adapted to our setting. As we have shown, some problems are as hard to approximate as any problem in ♯P, e.g., as hard as ♯CLIQUE. An interesting open problem is to find if there is a corresponding syntactic boundary of hardness: is it true that either a query is ♯BIS-easy or ♯CLIQUE-hard? We conjecture that such a syntactic boundary exists, though it remains open.

Compression and Approximation

There is long, successful line of work that compresses (deterministic) data to speed up query processing [54, 72, 76, 155, 166]. In wavelet approaches, probabilistic techniques are used to achieve a higher quality synopses, [54]. In contrast, lineage in our setting contains probabilities, which must be captured. The fact that the lineage is probabilistic raises the complexity of compression. For example, the approach of Garofalakis *et al.* [72] assumes that the entire wavelet transform can be computed efficiently. In our work, the transform size is exponential in the size of the data. Probabilistic query evaluation can be reduced to calculating a single coefficient of the transform, which implies exact computation of the transform is intractable [46, 78]. Aref *et al.* [60] advocate an approach to operate directly on compressed data to optimize queries on Biological sequences. However, this approach is not lineage aware and so cannot extract explanations from the compressed data.

In probabilistic databases, lineage is used for query processing in Mystiq [46,137] and Trio [168]. However, neither considers approximate lineage. Ré *et al.* [137] consider approximately computing the probability of a query answer, but do not consider the problem of storing the lineage of a query answer. These techniques are orthogonal: We can use the techniques of [137] to compute the top-k query probabilities from the Level II database using sufficient lineage. Approximate lineage is used to materialize views of probabilistic data; this problem has been previously considered [136], but only with an exact semantics.

Sen *et al.* [148] consider approximate processing of relational queries using graphical models, but not approximate lineage. In the graphical model literature [45,98] approximate representation is considered, where the goal is to compress the model for improved performance. However, the data and query models of the our approaches is different. Specifically, our approach leverages the fact that lineage is database is often *internal*.

Learning Theory

Our approach to computing polynomial lineage is based on computational learning techniques, such as the seminal paper by Linial *et al.* [114], and others, [19, 23, 125]. A key ingredient underlying these results are *switching lemmata*, [14, 87, 147]. For the problem of sufficient lineage, we use use

the implicit in both Segerlind *et al.* [147] and Trevisan [158] that either a few variables in a DNF matter (hit every clause) or the formula is ε large. The most famous (and sharpest) switching lemma due to Håstad [87] underlies the Fourier results. So far, learning techniques have only been applied to compressing the data, but have not compressed the lineage [12, 74]. A difference between our approach and this prior art is that we do not discard any tuples, but may discard lineage.

Explanation

Explanation is an important task for probabilistic databases that we only briefly touched on in Chapter 6. Explanation is a well-studied topic in the Artificial Intelligence community [86, 131]. The definition of explanation of a fact is a formula that is a minimal and sufficient to explain a fact – which is similar to our definition – but they additionally require that the formula be *unknown* to the user. We do not model the knowledge of users, but such a semantic would be very useful for scientists.

Chapter 8

CONCLUSION AND FUTURE WORK

This thesis demonstrates that it is possible to effectively manage large, imprecise databases using a generic approach based on probability theory. The technical contributions are two query-time techniques, *top-k query processing* and *aggregate evaluation*, and two view-based techniques: *materialized views* and *approximate lineage*. We demonstrated that a system, MystiQ, based on the techniques in this dissertation was able to support rich, structured queries on probabilistic databases that contain tens of gigabytes of data with performance comparable to a standard relational engine.

Future Work

The management of uncertainty will be an increasingly important area over the next several years as businesses, governments, and scientific researchers contend with an ever-expanding amount of data. Although the space of applications will be diverse, there will be fundemental primitives common to many of these applications (just as there with standard, deterministic data). As a result, there will be a ned for a general-purpose data management frameworks that can answer queries, explain results, and perform advanced analytics on large collections of imprecise data. The timing is right for such a system because of two complementary forces, a *technology push* and an *application pull*. The *technology push* is that there are a diverse and increasingly large set of technologies that produce imprecise data, such as entity matching, information extraction and inexpensive sensors. The *application pull* is the wide-variety of applications that require the ability to query, transform and process uncertain data, such as data integration, data exploration and preventive health-care monitoring applications. The critical problems of these future systems are of scale, performance and maintainability which are the cornerstones of the data management field.

An immediate challenge is to understand the trade-offs between the expressive power of probabilistic models and their ability to process large datasets. Consider an application that tracks thousands of users equipped with RFID sensors. Ideally, we would capture not only low-level physical

constraints, such as *"a user's current location is correlated with their previous location"*, but also higher-level correlation information, such as *"the database group has lunch together on Wednesday"*. An interesting question is: to what extent do specific types of correlations affect the output of a particular application? If our application only asks questions about groups of individuals, we could optimize our model to disregard low-level correlation information about any single individual. Dropping correlation information can vastly improve query performance both because we must process a much smaller amount of data, but also because we may be able to use more aggressive processing strategies. In the context of a database of RFID sensors, our preliminary results suggest that for some queries, not tracking correlations allows orders of magnitude performance improvement, e.g., we can process thousands more streams using the same resources. At the same, time there is only a small decrease in quality, e.g., our detection rates for events decreases only slightly. There are many other opportunities for aggressive approximations in probabilistic query processing. Approximation techniques for query processing will be crucial in probabilistic database applications, but our understanding of its limits is still in its infancy.

Long-term work Current database management systems are well-suited to informing users of the who, what, and where of data processing, e.g., *"which of my stores has a projected profit?"*, but do a poor job of informing users about the *why* and *how* of data processing, e.g., *"why does store six have a projected profit?"*. Worse still, the next generation of data products, such as forecasting data, similarity scores or information extraction tools; are *less precise* than traditional relational data and so, *more difficult to understand*. A transparent database would allow me to tackle current data management problems, such as *explaining the provenance of data*, but also *emerging data management problems*, such as debugging the output of information extraction tools. For example, consider the database of a large retailer that contains standard relational data, such as current inventory levels and orders, and also contains imprecise data, such as the result of forecasting software. In response to the query above, the system would return an *explanation* such as *"we predict a large profit in store six because predictive model 10 says that cameras will sell well in the Southwest where store 10 is located*." A facility for explanations is the necessary primitive to build a system that allows an analyst to interactively explore and understand a large collection of imprecise data. A transparent database could also support *what-if analyses* where our goal is to understand how changes in the

underlying data affect the final output. Adapting existing notions of explanations from the artificial intelligence community [29, 131] to the problem of explaining the results of complex queries on large-scale data products is a major open challenge.

BIBLIOGRAPHY

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *PODS*, pages 254–263, 1998.

[3] Serge Abiteboul and Pierre Senellart. Querying and updating probabilistic information in xml. In *EDBT*, pages 1059–1068, 2006.

[4] Raman Adaikkalavan and Sharma Chakravarthy. Snoopib: Interval-based event specification and detection for active databases. *Data Knowl. Eng.*, 59(1):139–165, 2006.

[5] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. Automated selection of materialized views and indexes in sql databases. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 496–505. Morgan Kaufmann, 2000.

[6] Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pages 586–596, 2002.

[7] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases. In *ICDE*, 2006.

[8] L. Antova, C. Koch, and D. Olteanu. World-set decompositions: Expressiveness and efficient algorithms. In *ICDT*, pages 194–208, 2007.

[9] Lyublena Antova, Christoph Koch, and Dan Olteanu. 10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information. In *ICDE*, pages 606–615, 2007.

[10] Arvind Arasu, Christopher Ré, and Dan Suciu. Large-scale deduplication with constraints using Dedupalog (full research paper). In *ICDE*, 2009. *to appear*.

[11] Marcelo Arenas, Leopoldo E. Bertossi, Jan Chomicki, Xin He, Vijay Raghavan, and Jeremy Spinrad. Scalar aggregation in inconsistent databases. *Theor. Comput. Sci.*, 296(3):405–434, 2003.

[12] S. Babu, M. Garofalakis, and R. Rastogi. Spartan: A model-based semantic compression system for massive data tables. In *SIGMOD*, pages 283–294, 2001.

[13] D. Barbara, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 4(5):487–502, 1992.

[14] P. Beame. A switching lemma primer. Technical Report 95-07-01, University of Washington, Seattle, WA, 1995.

[15] O. Benjelloun, A. Das Sarma, A. Y. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *VLDB J.*, 17(2):243–264, 2008.

[16] O. Benjelloun, A. Das Sarma, C. Hayworth, and J. Widom. An introduction to ULDBs and the Trio system. *IEEE Data Eng. Bull*, 29(1):5–16, 2006.

[17] Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, and Jennifer Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.

[18] L. Bertossi and J. Chomicki. Query answering in inconsistent databases. In G. Saake J. Chomicki and R. van der Meyden, editors, *Logics for Emerging Applications of Databases*. Springer, 2003.

[19] A. Blum, M. L. Furst, J C. Jackson, M J. Kearns, Y. Mansour, and S. Rudich. Weakly learning dnf and characterizing statistical query learning using fourier analysis. In *STOC*, pages 253–262, 1994.

[20] B. Boeckmann, A. Bairoch, R. Apweiler, M. C. Blatter, A. Estreicher, E. Gasteiger, M. J. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider. The swiss-prot protein knowledgebase and its supplement trembl in 2003. *Nucleic Acids Res*, 31(1):365–370, January 2003.

[21] Jihad Boulos, Nilesh N. Dalvi, Bhushan Mandhani, Shobhit Mathur, Christopher Ré, and Dan Suciu. Mystiq: a system for finding more answers by using probabilities (demonstration). In Fatma Özcan, editor, *SIGMOD Conference*, pages 891–893. ACM, 2005.

[22] Lars Brenna, Alan J. Demers, Johannes Gehrke, Mingsheng Hong, Joel Ossher, Biswanath Panda, Mirek Riedewald, Mohit Thatte, and Walker M. White. Cayuga: a high-performance event processing engine. In *SIGMOD Conference*, pages 1100–1102, 2007.

[23] N. Bshouty and C. Tamon. On the fourier spectrum of monotone functions. *J. ACM*, 43(4):747–770, 1996.

[24] P. Buneman, A. Chapman, and J. Cheney. Provenance management in curated databases. In *SIGMOD*, pages 539–550, 2006.

[25] Peter Buneman, James Cheney, Wang Chiew Tan, and Stijn Vansummeren. Curated databases. In *PODS*, pages 1–12, 2008.

[26] D. Burdick, P. M. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. Olap over uncertain and imprecise data. *VLDB J.*, 16(1):123–144, 2007.

[27] Douglas Burdick, Prasad Deshpande, T. S. Jayram, Raghu Ramakrishnan, and Shivakumar Vaithyanathan. Olap over uncertain and imprecise data. In *VLDB*, pages 970–981, 2005.

[28] Roger Cavallo and Michael Pittarelli. The theory of probabilistic databases. In *Proceedings of VLDB*, pages 71–81, 1987.

[29] Urszula Chajewska and Joseph Y. Halpern. Defining explanation in probabilistic systems. In Dan Geiger and Prakash P. Shenoy, editors, *UAI*, pages 62–71. Morgan Kaufmann, 1997.

[30] A. Chapman and H. V. Jagadish. Issues in building practical provenance systems. *IEEE Data Eng. Bull.*, 30(4):38–43, 2007.

[31] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *ACM SIGMOD*, San Diego, CA, 2003.

[32] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, pages 313–324, 2003.

[33] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. Optimizing queries with materialized views. In *ICDE*, pages 190–200, 1995.

[34] R. Cheng, D. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proc. of SIGMOD03*, 2003.

[35] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Incorporating constraints in probabilistic xml. In *PODS*, pages 109–118, 2008.

[36] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Running tree automata on probabilistic xml. In *PODS*, pages 227–236, 2009.

[37] The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. In *Nature Genet.*, pages 25–29, (2000).

[38] Graham Cormode, Feifei Li, and Ke Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE*, pages 305–316, 2009.

[39] Garage Band Corp. `http://www.garageband.com/`.

[40] Garage Band Corp. `www.ilike.com`.

[41] Microsoft Corp. Northwind for sql server 2000.

[42] Microsoft Corp. Sql server 2005 samples (feb. 2007).

[43] Transaction Processing Performance Council. Tpc-h (ad-hoc, decision support) benchmark. http://www.tpc.org/.

[44] Transaction Processing Performance Council. Tpc-r (decision support) benchmark (obsolete). http://www.tpc.org/.

[45] R. G. Cowell, S. L. Lauritzen, A. P. David, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[46] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, Toronto, Canada, 2004.

[47] N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.

[48] N. Dalvi and D. Suciu. Management of probabilisitic data: Foundations and challenges. In *PODS*, pages 1–12, 2007.

[49] Nilesh Dalvi, Christopher Ré, and Dan Suciu. Queries and materialized views on probabilistic databases. *JCSS*, 2009.

[50] Adnan Darwiche. A differential approach to inference in bayesian networks. *J. ACM*, 50(3):280–305, 2003.

[51] A. Das Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.

[52] Rodrigo de Salvo Braz, Eyal Amir, and Dan Roth. Lifted first-order probabilistic inference. In *IJCAI*, pages 1319–1325, 2005.

[53] Special issue on data provenance. *IEEE Data Eng. Bull.*, 30(4), 2007.

[54] A. Deligiannakis, M. Garofalakis, and N. Roussopoulos. Extended wavelets for multiple measures. *ACM Trans. Database Syst.*, 32(2):10, 2007.

[55] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004.

[56] Amol Deshpande, Carlos Guestrin, Samuel Madden, Joseph M. Hellerstein, and Wei Hong. Model-driven data acquisition in sensor networks. In M.A. Nascimento, M.T. Özsu, D. Kossmann, R.J. Miller, J.A. Blakeley, and K.B. Schiefer, editors, *VLDB*, pages 588–599. Morgan Kaufmann, 2004.

[57] Debabrata Dey and Sumit Sarkar. Generalized normal forms for probabilistic relational data. *IEEE Trans. Knowl. Data Eng.*, 14(3):485–497, 2002.

[58] Martin E. Dyer, Leslie Ann Goldberg, Catherine S. Greenhill, and Mark Jerrum. On the relative complexity of approximate counting problems. In *APPROX*, pages 108–119, 2000.

[59] Martin E. Dyer, Leslie Ann Goldberg, and Mark Jerrum. An approximation trichotomy for boolean #csp. *CoRR*, abs/0710.4272, 2007.

[60] M. Eltabakh, M. Ouzzani, and W. G. Aref. bdbms - a database management system for biological data. In *CIDR*, pages 196–206, 2007.

[61] Herbert B. Enderton. *A Mathematical Introduction To Logic*. Academic Press, San Diego, 1972.

[62] Oren Etzioni, Michele Banko, and Michael J. Cafarella. Machine reading. In *AAAI*, 2006.

[63] Ronald Fagin and Joseph Y. Halpern. Reasoning about knowledge and probability. In Moshe Y. Vardi, editor, *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 277–293, San Francisco, 1988. Morgan Kaufmann.

[64] Ronald Fagin, Joseph Y. Halpern, and Nimrod Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87(1/2):78–128, 1990.

[65] I. P. Fellegi and A. B. Sunter. A theory for record linkage. In *Journal of the American Statistical Society*, volume 64, pages 1183–1210, 1969.

[66] `http://flybase.bio.indiana.edu/`.

[67] N. Friedman, L .Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI*, pages 1300–1309, 1999.

[68] Norbert Fuhr and Thomas Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[69] A. Fuxman and R. J. Miller. First-order query rewriting for inconsistent databases. In *ICDT*, pages 337–351, 2005.

[70] Ariel Fuxman, Elham Fazli, and Renée J. Miller. Conquer: Efficient management of inconsistent databases. In *SIGMOD Conference*, pages 155–166, 2005.

[71] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian-Augustin Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pages 371–380, 2001.

[72] M. Garofalakis and P. Gibbons. Probabilistic wavelet synopses. *ACM Trans. Database Syst.*, 29:43–90, 2004.

[73] Tingjian Ge, Stanley B. Zdonik, and Samuel Madden. Top-*k* queries on uncertain data: on score distribution and typical answers. In *SIGMOD Conference*, pages 375–388, 2009.

[74] L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. In *SIGMOD*, pages 461–472, 2001.

[75] J. Goldstein and P. Larson. Optimizing queries using materialized views: a practical, scalable solution. In *SIGMOD 2001*, pages 331–342, New York, NY, USA, 2001. ACM Press.

[76] J. Goldstein, R. Ramakrishnan, and U. Shaft. Compressing relations and indexes. In *ICDE*, pages 370–379, 1998.

[77] S.M. Gorski, S. Chittaranjan, E.D. Pleasance, J.D. Freeman, C.L. Anderson, R.J. Varhol, S.M. Coughlin, S.D. Zuyderduyn, S.J. Jones, and M.A. Marra. A SAGE approach to discovery of genes involved in autophagic cell death. *Curr. Biol.*, 13:358–363, Feb 2003.

[78] Erich Grädel, Yuri Gurevich, and Colin Hirsch. The complexity of query reliability. In *PODS*, pages 227–234, 1998.

[79] G. Grahne. Lncs 554: The problem of incomplete information in relational databases. 1991.

[80] T. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. G. Ives, and V. Tannen. Orchestra: facilitating collaborative data sharing. In *SIGMOD*, pages 1131–1133, 2007.

[81] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[82] Todd Green and Val Tannen. Models for incomplete and probabilistic information. *IEEE Data Engineering Bulletin*, 29(1):17–24, 2006.

[83] Lifang Gu, Rohan Baxter, Deanne Vickers, and Chris Rainsford. Record linkage: Current practice and future directions. In *CMIS Technical Report No. 03/83*, 2003.

[84] R. Gupta and S. Sarawagi. Curating probabilistic databases from information extraction models. In *Proc. of the 32nd Int'l Conference on Very Large Databases (VLDB)*, 2006.

[85] Alon Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[86] J. Halpern and J. Pearl. Causes and explanations: A structural-model approach - part II: Explanations. In *IJCAI*, pages 27–34, 2001.

[87] J. Håstad. *Computational limitations for small depth circuits*. M.I.T Press, Cambridge, Massachusetts, 1986.

[88] E. Hazan, S. Safra, and O. Schwartz. On the hardness of approximating k-dimensional matching. *ECCC*, 10(020), 2003.

[89] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.

[90] Ming Hua, Jian Pei, Wenjie Zhang, and Xuemin Lin. Efficiently answering probabilistic threshold top-k queries on uncertain data. In *ICDE*, pages 1403–1405, 2008.

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

[92] Edward Hung, Lise Getoor, and V. S. Subrahmanian. Pxml: A probabilistic semistructured data model and algebra. In *ICDE*, pages 467–, 2003.

[93] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31:761–791, October 1984.

[94] ISO. *Standard 9075. Information Processing Systems. Database Language SQL*, 1987.

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

[96] T. S. Jayram, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and Huaiyu Zhu. Avatar information extraction system. *IEEE Data Eng. Bull.*, 29(1):40–48, 2006.

[97] T.S. Jayram, S. Kale, and E. Vee. Efficient aggregation algorithms for probabilistic data. In *SODA*, 2007.

[98] M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

[99] Bhargav Kanagal and Amol Deshpande. Online filtering, smoothing and probabilistic modeling of streaming data. In *ICDE*, pages 1160–1169, 2008.

[100] Bhargav Kanagal and Amol Deshpande. Indexing correlated probabilistic databases. In *SIGMOD Conference*, pages 455–468, 2009.

[101] Richard Karp and Michael Luby. Monte-carlo algorithms for enumeration and reliability problems. In *STOC*, 1983.

[102] Richard M. Karp and Michael Luby. Monte-carlo algorithms for enumeration and reliability problems. In *FOCS*, pages 56–64, 1983.

[103] Benny Kimelfeld. *Querying Paradigms for the Web*. PhD thesis, The Hebrew University, August 2008.

[104] Benny Kimelfeld, Yuri Kosharovsky, and Yehoshua Sagiv. Query efficiency in probabilistic xml models. In *SIGMOD Conference*, pages 701–714, 2008.

[105] Christoph Koch. Approximating predicates and expressive queries on probabilistic databases. In *PODS*, pages 99–108, 2008.

[106] Christoph Koch. A compositional query algebra for second-order logic and uncertain databases. In *ICDT*, pages 127–140, 2009.

[107] E. Kushilevitz and Y. Mansour. Learning decision trees using the fourier spectrum. *SIAM J. Comput.*, 22(6):1331–1348, 1993.

[108] L. Lakshmanan, N. Leone, R. Ross, and V.S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3), 1997.

[109] Serge Lang. *Algebra*. Springer, January 2002.

[110] J. Lester, T. Choudhury, N. Kern, G. Borriello, and B. Hannaford. A hybrid discriminative/generative approach for modeling human activities. In *IJCAI*, pages 766–772, 2005.

[111] Julie Letchner, Christopher Ré, Magdalena Balazinska, and Mathai Philipose. Lahar demonstration: Warehousing markovian streams. In *VLDB*, 2009.

[112] Julie Letchner, Christopher Ré, Magdalena Balazinska, and Matthai Philipose. Access methods for markovian streams. In *ICDE*, pages 246–257, 2009.

[113] Jian Li, Barna Saha, and Amol Deshpande. A unified approach to ranking in probabilistic databases. In *VLDB*, 2007.

[114] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993.

[115] A. Machanavajjhala and J .Gehrke. On the efficiency of checking perfect privacy. In Stijn Vansummeren, editor, *PODS*, pages 163–172. ACM, 2006.

[116] Eirinaios Michelakis, Rajasekar Krishnamurthy, Peter J. Haas, and Shivakumar Vaithyanathan. Uncertainty management in rule-based information extraction systems. In *SIGMOD Conference*, pages 101–114, 2009.

[117] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *SIGMOD*, 2004.

[118] Gerome Miklau. *Confidentiality and Integrity in Data Exchange*. PhD thesis, University of Washington, Aug 2005.

[119] Gerome Miklau and Dan Suciu. A formal analysis of information disclosure in data exchange. *J. Comput. Syst. Sci.*, 73(3):507–534, 2007.

[120] Katherine F. Moore, Vihbor Rasogi, Christopher Ré, and Dan Suciu. Query containment of tier-2 queries over a probabilsitic database. In *Managment of Uncertain Databases*, 2009.

[121] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1997.

[122] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo I. Seltzer. Provenance-aware storage systems. In *USENIX Annual Technical Conference, General Track*, pages 43–56, 2006.

[123] Raghotham Murthy, Robert Ikeda, and Jennifer Widom. Making aggregation work in uncertain and probabilistic databases. Technical Report 2007-7, Stanford InfoLab, June 2007.

[124] Andrew Nierman and H. V. Jagadish. Protdb: Probabilistic data in xml. In *VLDB*, pages 646–657, 2002.

[125] R. W. O'Donnell. *Computational Applications of Noise Sensitivity*. PhD thesis, M.I.T., 2003.

[126] O.Etzioni, M.J. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D.S. Weld, and A. Yates. Web-scale information extraction in knowitall: (preliminary results). In S.I. Feldman, M. Uretsky, M. Najork, and C.E. Wills, editors, *WWW*, pages 100–110. ACM, 2004.

[127] Dan Olteanu, Jiewen Huang, and Christoph Koch. SPROUT: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *Proc. of ICDE 2009*, 2009.

[128] Christos Papadimitriou. *Computational Complexity*. Addison Wesley Publishing Company, 1994.

[129] A. Parag, O. Benjelloun, A.D. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom. Trio: A system for data uncertainty and lineage. In *VLDB*, 2006.

[130] J. Pearl. *Probabilistic Reasoning In Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc, 1988.

[131] Judea Pearl. *Causality : Models, Reasoning, and Inference*. Cambridge University Press, March 2000.

[132] Jian Pei, Bin Jiang, Xuemin Lin, and Yidong Yuan. Probabilistic skylines on uncertain data. In *VLDB*, pages 15–26, 2007.

[133] Rachel Pottinger and Alon Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB J.*, 10(2-3):182–198, 2001.

[134] `http://www.pubmed.gov`.

[135] C. Ré, N. Dalvi, and D. Suciu. Query evaluation on probabilistic databases. *IEEE Data Engineering Bulletin*, 29(1):25–31, 2006.

[136] C. Ré and D. Suciu. Materialized views in probabilistic databases for information exchange and query optimization. In *VLDB*, pages 51–62, 2007.

[137] Christopher Ré, Nilesh N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data (full research paper). In *ICDE*, pages 886–895. IEEE, 2007.

[138] Christopher Ré, Julie Letchner, Magdalena Balazinska, and Dan Suciu. Event queries on correlated probabilistic streams. In *SIGMOD Conference*, pages 715–728, 2008.

[139] Christopher Ré and Dan Suciu. Efficient evaluation of `HAVING` queries. In *DBPL*, pages 186–200, 2007.

[140] Christopher Ré and Dan Suciu. Materialized views in probabilistic databases for information exchange and query optimization. In *VLDB*, pages 51–62, 2007.

[141] Christopher Ré and Dan Suciu. Approximate lineage for probabilistic databases. *(Formally, VLDB) PVLDB*, 1(1):797–808, 2008.

[142] Christopher Ré and Dan Suciu. Managing probabilistic data with Mystiq: The can-do, the could-do, and the can't-do. In *SUM*, pages 5–18, 2008.

[143] Christopher Ré and Dan Suciu. The trichotomy of `HAVING` queries on a probabilistic database. *VLDB Journal*, 2009.

[144] Robert Ross, V. S. Subrahmanian, and John Grant. Aggregate operators in probabilistic databases. *J. ACM*, 52(1):54–101, 2005.

[145] A.D. Sarma, O. Benjelloun, A.Y. Halevy, and J. Widom. Working models for uncertain data. In Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, *ICDE*, page 7. IEEE Computer Society, 2006.

[146] Anish Das Sarma, Martin Theobald, and Jennifer Widom. Exploiting lineage for confidence computation in uncertain and probabilistic databases. In *ICDE*, pages 1023–1032, 2008.

[147] N. Segerlind, S. R. Buss, and R. Impagliazzo. A switching lemma for small restrictions and lower bounds for k-dnf resolution. *SIAM J. Comput.*, 33(5):1171–1200, 2004.

[148] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *Proceedings of ICDE*, 2007.

[149] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Exploiting shared correlations in probabilistic databases. *PVLDB*, 1(1):809–820, 2008.

[150] Pierre Senellart and Serge Abiteboul. On the complexity of managing probabilistic xml data. In *PODS*, pages 283–292, 2007.

[151] R. Servedio. On learning monotone dnf under product distributions. *Inf. Comput.*, 193(1):57–74, 2004.

[152] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Inf. Comput.*, 82(1):93–133, 1989.

[153] Parag Singla and Pedro Domingos. Lifted first-order belief propagation. In *AAAI*, pages 1094–1099, 2008.

[154] M. Soliman, I.F. Ilyas, and K. Chen-Chaun Chang. Top-k query processing in uncertain databases. In *Proceedings of ICDE*, 2007.

[155] M. Stonebraker, D. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-store: A column-oriented dbms. In *VLDB*, pages 553–564, 2005.

[156] Go database v. go200801.

[157] Thanh Tran, Charles Sutton, Richard Cocci, Yanming Nie, Yanlei Diao, and Prashant J. Shenoy. Probabilistic inference over rfid streams in mobile environments. In *ICDE*, pages 1096–1107, 2009.

[158] L. Trevisan. A note on deterministic approximate counting for k-dnf. *Electronic Colloquium on Computational Complexity (ECCC)*, (069), 2002.

[159] Jeffrey D. Ullman. *Principles of Database and Knowledgebase Systems I*. Computer Science Press, Rockville, MD 20850, 1989.

[160] Jeffrey D. Ullman. Information integration using logical views. In Foto N. Afrati and Phokion G. Kolaitis, editors, *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 1997.

[161] Patrick Valduriez. Join indices. *ACM Trans. Database Syst.*, 12(2):218–246, 1987.

[162] L.G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

[163] Maurice van Keulen, Ander de Keijzer, and Wouter Alink. A probabilistic xml approach to data integration. In *ICDE*, pages 459–470, 2005.

[164] M. Y. Vardi. The complexity of relational query languages. In *Proceedings of 14th ACM SIGACT Symposium on the Theory of Computing*, pages 137–146, San Francisco, California, 1982.

[165] M. Y. Vardi. On the integrity of databases with incomplete information. In *Proceedings of 5th ACM Symposium on Principles of Database Systems*, pages 252–266, 1986.

[166] J. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD*, pages 193–204, 1999.

[167] Daisy Zhe Wang, Eirinaios Michelakis, Minos N. Garofalakis, and Joseph M. Hellerstein. Bayesstore: managing large, uncertain data repositories with probabilistic graphical models. *PVLDB*, 1(1):340–351, 2008.

[168] Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.

[169] Stephen E. Fienberg William W. Cohen, Pradeep Ravikumar. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.

[170] William Winkler. The state of record linkage and current research problems. In *Technical Report, Statistical Research Division, U.S. Bureau of the Census*, 1999.

[171] Ke Yi, Feifei Li, George Kollios, and Divesh Srivastava. Efficient processing of top-k queries in uncertain databases. In *ICDE*, pages 1406–1408, 2008.

[172] Xi Zhang and Jan Chomicki. On the semantics and evaluation of top-k queries in probabilistic databases. In *ICDE Workshops*, pages 556–563, 2008.

[173] http://imdb.com.

[174] http://www.amazon.com.

Appendix A

PROOF OF RELAXED PROGRESS

A.1 Bounding the Violations of Progress

The high level goal of this section is to establish that after a negligible number of steps with respect to the optimal, the progress assumption holds. The result contained here, allows us to show that with extremely high probability our algorithm takes $2(OPT + o(OPT))$ steps, thus achieving our stated competitive ratio of 2.

To achieve this goal, our technical tool is an exponentially small upper bound on the probability that progress is violated. Further, we do not assume anything about the underlying distribution of true probabilities, thus keeping our competitive result free of distributional assumptions. Specifically, in what follows, all random choices are on the results of the experiment with the underlying probabilities chosen adversarial.

We first introduce some notation and define our random variables, an estimation lemma and prove our main result.

A.1.1 Notation and Random Variables

We let n be the number of iterations on the current interval, and consider for a fixed δ what is the probability that progress is violated after we take n_0 steps. We denote the (fixed) true probability p.

Interval Size Function Let $f(n, \delta)$ be the bounding function,

$$f(n, \delta) = \frac{2m}{\sqrt{n}} \log \frac{2}{\delta}$$

Random Variables We have two main random variables, $\rho(n)$ and $X(n_0)$. $\rho(n)$ represents the value of our estimator after n draws. $X(n_0)$ represents the value of n_0 random draws after the n. We observe that all though we have described them as sequential, they are actually independent random

variables because they are sums of independent trials. We summarize the properties of these random variables below.

r.v. α	$\mathbf{E}[\alpha]$	As Sums of Indicators
$\rho(n)$	p	$\frac{1}{n}\sum_{i=1}^{n_0} y_i$
$X(n_0)$	$n_0 p$	$\sum_{j=1}^{n} x_j$

Notation An obvious extension is $\rho(n+n_0)$, the estimate after $n+n_0$ steps. This random variable can be written in terms of the other two (independent) rvs as,

$$\rho(n+n_0) = \frac{\rho(n) * n + X(n_0)}{n+n_0} \tag{A.1}$$

And the following decompositions are helpful later

$$\rho(n+n_0) - \rho(n) = \frac{X(n_0) - \rho(n)n_0}{n+n_0} \tag{A.2}$$

and,

$$\rho(n) - \rho(n+n_0) = \frac{\rho(n)n_0 - X(n_0)}{n+n_0} \tag{A.3}$$

Progress Conditions Progress is violated when either of two conditions are met

1. $\rho(n) < \rho(n+n_0)$ and
$$\rho(n) + f(n,\delta) < \rho(n+n_0) + f(n+n_0,\delta')$$

2. $\rho(n) > \rho(n+n_0)$ and
$$\rho(n) - f(n,\delta) > \rho(n+n_0) - f(n+n_0,\delta')$$

A.1.2 Helpful Lemma

Lemma A.1.1. *Let $n_0 = n^\alpha$ and β be a constant such that $\beta > 1$. Then the following holds,*

$$\frac{1}{\sqrt{n}} - \frac{1}{\sqrt{n+n_0}} \in \Theta(n^{\alpha-3/2}) \tag{A.4}$$

Proof. We first factor out $n^{-1/2}$ and rearrange the second term to get the inequality

$$cn^{\alpha-1} \leq 1 - (\frac{n}{n+n_0})^{1/2} \leq cn^{\alpha-1}$$

We prove that $g(n) = 1 - (\frac{n}{n+1})^{1/2} \in \Theta(n^{-1})$. We observe that if $n_0 = n^\alpha$, factoring will yield $g(n^{1-\alpha})$. Using composition we get that $g(n^{1-\alpha}) = \Theta(n^{\alpha-1})$, which implies our lemma.

Written another way, we are comparing $1 - \frac{1}{n} = \frac{n-1}{n}$ and $\sqrt{\frac{n}{n+1}}$.

$$\frac{(n-1)^2}{n^2} \propto \frac{n}{n+1} \Leftrightarrow (n-1)^2(n+1) \propto n^3$$

□

A.1.3 Chernoff Version of Bounds

The idea here is to exponentiate the bounds, i.e. (e^{tX}) and then apply a Markov inequality. We then optimize our choice of t to get a minimize the probability.

$$\begin{aligned}
\mathbf{P}[\text{Progress is violated}](n, n_0, \delta) \leq \quad & \mathbf{P}[(f(n,\delta) - f(n+n_0,\delta)) < (\rho(n+n_0) - \rho(n))] + \\
& \mathbf{P}[(f(n,\delta) - f(n+n_0,\delta)) < (\rho(n) - \rho(n+n_0))] && \text{Union Bound} \\
\leq \quad & 2\mathbf{P}[-e^{(f(n,\delta)-f(n+n_0,\delta))} < e^{(\rho(n+n_0)-\rho(n))^2}] && \text{Dual is the same} \\
\leq \quad & 2\mathbf{E}[e^{(\rho(n+n_0)-\rho(n))}]e^{-(f(n,\delta)-f(n+n_0,\delta))} && \text{Markov after } e^t
\end{aligned}$$

The dual case is $\leq \mathbf{E}[e^{t(\rho(n)-\rho(n+n_0))}]e^{-t(f(n,\delta)-f(n+n_0,\delta))}$. We will see that it is no worse than the case we have chosen.

The Expectation term

Lemma A.1.2. *If $t << n + n_0$ then $\mathbf{E}[e^{t(\rho(n+n_0)-\rho(n))}]$ and $\mathbf{E}[e^{t(\rho(n)-\rho(n+n_0))}]$ are both $O(1)$.*

Proof. We do the calculation for $\mathbf{E}[e^{t(\rho(n+n_0)-\rho(n))}]$,

$$\begin{aligned}\mathbf{E}[e^{t(\rho(n+n_0)-\rho(n))}] = & \quad \mathbf{E}[e^{t(\frac{X(n_0)}{n+n_0}-\frac{n_0\rho(n)}{n+n_0})}] & \text{Decomposition in Eq.. A.2}\\ = & \quad \mathbf{E}[e^{\frac{tX(n_0)}{n+n_0}}]\mathbf{E}[e^{-\frac{tn_0\rho(n)}{n+n_0}}] & \text{independence of } X(n_0), \rho(n)\end{aligned}$$

Thus we only need to establish that if $t << n + n_0$ then

1. $\mathbf{E}[e^{\frac{tX(n_0)}{n+n_0}}] \in O(1)$
2. $\mathbf{E}[e^{\frac{tn_0\rho(n)}{n+n_0}}] \in O(1)$

First term: $\mathbf{E}[e^{\frac{tX(n_0)}{n+n_0}}]$

$$\begin{aligned}\mathbf{E}[e^{\frac{tX(n_0)}{n+n_0}}] = & \quad \mathbf{E}[e^{\sum_{i=1}^{n_0}\frac{tx_i}{n+n_0}}] & \text{Definition of } X(n_0)\\ = & \quad \prod_{i=1}^{n_0}\mathbf{E}[e^{\frac{tx_i}{n+n_0}}] & \text{independence of } \{x_i\}\\ = & \quad \prod_{i=1}^{n_0}(e^{\frac{t}{n+n_0}}p + 1 - p) & \text{Expectation}\\ = & \quad \prod_{i=1}^{n_0}((e^{\frac{t}{n+n_0}} - 1)p + 1) & \text{factoring } p\\ < & \quad \prod_{i=1}^{n_0} e^{e^{\frac{t}{n+n_0}}-1}p & 1 + x < e^x\\ = & \quad e^{n_0 p(e^{\frac{t}{n+n_0}}-1)} & \text{exp rules}\end{aligned}$$

We observe that for $t << n + n_0$, then this term goes to $e^0 = 1$ as desired.

Second Term: $\mathbf{E}[e^{\frac{tn_0\rho(n)}{n+n_0}}]$ We deal with the second expectation term.

$$\begin{aligned}\mathbf{E}[e^{\rho(n)\frac{t}{(n+n_0)}}] = & \quad \mathbf{E}[e^{\sum_{i=1}^{n} y_i\frac{t}{n(n+n_0)}}]\\ = & \quad exp(np(e^{\frac{t}{n(n+n_0)}} - 1)) & \text{Same argument}\end{aligned}$$

Now, we observe that for $t << n(n + n_0)$ this term will also go to 1. □

The Denominator

Lemma A.1.3. *For any choice of t,*

$$e^{-t(f(n,\delta)-f(n+n_0,\delta))} \leq e^{2tm\log\frac{2}{\delta}n^{\alpha-3/2}}$$

Proof.

$$\begin{aligned} e^{-t(f(n,\delta)-f(n+n_0,\delta))} &= e^{2tm\log\frac{2}{\delta}(n^{-1/2}-(n+n_0)^{-1/2})} \\ &\leq e^{2mt\log\frac{2}{\delta}n^{\alpha-3/2}} \qquad \text{Lemma } A.1.1 \end{aligned}$$

□

The Punchline

Combining lemma A.1.3 and A.1.2, we have

$$t << n + n_0 \implies \mathbf{P}[\text{Progress is violated}](n, n_0, \delta) \leq 2e^{2mt\log\frac{2}{\delta}n^{\alpha-3/2}}$$

We take $t = n_0 = n^\alpha$ for $\alpha < 1$, which satisfies $t << n + n_0$.

$$\mathbf{P}[\text{Progress is violated}](n, n_0, \delta) \leq 2e^{2m\log\frac{2}{\delta}n^{2\alpha-3/2}}$$

If we take $t = n^\alpha = \frac{3}{4} + \alpha'$, for $\alpha' \in [0, \frac{1}{4})$ then our probability simplifies to:

$$\mathbf{P}[\text{violation}](n, \alpha', \delta) \leq 2e^{-2m\log\frac{2}{\delta}n^{\alpha'}} \tag{A.5}$$

Since this is exponentially small probability, the number and variance of errors are exponentially small. Thus by applying a Chebyshev inequality we can conclude with exponentially high

probability there are no errors after n^{α} steps from any fixed n.

Theorem A.1.4. *The probability that progress holds after some negligible number of steps tends to* 1 *exponentially quickly.*

Remark A.1.5. *Also as long as* $\alpha < 1$*, our optimality ratio is preserved.*

Appendix B

PROOFS FOR EXTENSIONAL EVALUATION

B.1 Properties of Safe Plans

We formalize the properties that hold in safe extensional plans in the following proposition:

Proposition B.1.1. *Let $P = \pi^I_{-x} P_1$ be a safe plan then for any tuples $t_1, \ldots, t_n \in \mathbb{D}^{|\mathbf{var}(P_1)|}$ that disagree on x, i.e., such that $i \neq j$ implies that $t_i[\mathbf{var}(P)] = t_j[\mathbf{var}(P)]$ and $t_i[x] \neq t_j[x]$ and then for any $s_1, \ldots, s_n \in S$ we have independence, that is the following equation holds:*

$$\mu\left(\bigwedge_{i=1,\ldots,n} \omega_{P_1,S}(t_i) = s_i\right) = \prod_{i=1,\ldots,n} \mu\left(\omega_{P_1,S}(t_i) = s_i\right) \tag{B.1}$$

Similarily, let $P = \pi^D P_1$ then we have disjointness:

$$\mu\left(\bigwedge_{i=1,\ldots,n} \omega_{P_1,S}(t_i) = 0\right) = \sum_{i=1,\ldots,n} \mu\left(\omega_{P_1,S}(t_i) = 0\right) - (n-1) \tag{B.2}$$

Let $P = P_1 \bowtie P_2$, then for any tuples $t_i \in \mathbb{D}^{|\mathbf{var}(P_i)|}$ for $i = 1, 2$ then and $s_1, s_2 \in S$, we have independence:

$$\mu\left(\omega_{P_1,S}(t_1) = s_1 \wedge \omega_{P_1,S}(t_2) = s_2\right) = \mu\left(\omega_{P_1,S}(t_1) = s_1\right)\mu\left(\omega_{P_1,S}(t_2) = s_2\right) \tag{B.3}$$

Proof. We prove Eq. B.1. To see this observe that, directly from the definition, the set of tuples that contribute to t_i and t_j ($i \neq j$) do not share the same value for a key in *any* relation. It is not hard to see that t_i and t_j are functions of independent tuples, hence are independent. The equation then follows by definition of independence.

We prove Eq. B.2. Assume for contradiction that the tuples are not disjoint, that is there exists some possible world W such that for some $i \neq j$ $\left\{t_i, t_j\right\} \subseteq W$. By the definition, there must exist some

key goal g such that $\mathbf{key}(g) \subseteq \mathbf{var}(P)$. Thus, for t_i and t_j to be present in W it must be that there are two distinct tuples with the same key value – but different values for the attribute corresponding to x. This is a contradiction to the key condition, hence the tuples are disjoint and the equation follows.

We prove Eq. B.3. In a safe plan, $\mathbf{goal}(P_1) \cap \mathbf{goal}(P_2) = \emptyset$ and distinct relations are independent. As a result, the tuples themselves are independent. □

B.2 Appendix: Full Proof for `COUNT(DISTINCT)`

Theorem B.2.1 (Restatement of Theorem 4.4.14)**.** *Let Q be `COUNT(DISTINCT)`-safe then its evaluation problem is in* P.

Proof. Since Q is `COUNT(DISTINCT)`-safe, then there is a safe plan P for the skeleton of Q. In the following let $P_1 \prec P_2$ denote the relationship that P_1 is a descendant in P of P_2 (alternatively, containment). Let P_y be a subplan which satisfies $P_{-y} = \pi^I_{-y}(P') \prec P$ or $P_{-y} = \pi^D_{-y}(P') \prec P$. P_{-y} is a safe plan, hence S-safe for $S = \mathbb{Z}_2$, i.e., the `EXISTS` algebra. For each t, we can write $\hat{\omega}^I_{P_{-y}}(t) = (1-p, p)$, i.e., t is present with probability p. From this, create a marginal vector in $\mathbb{Z}^{k+1}$, as in `COUNT`, $\boldsymbol{m}^t$ such that $\boldsymbol{m}^t[0] = 1 - p$ and $\boldsymbol{m}^t[1] = p$ and all other entries 0. Notice that if $t \neq t'$ then $t[y] \neq t[y']$. Informally, this means all y values are distinct "after" P_y.

Compute the remainder of P as follows: If P_0 is not a proper ancestor or descendant of P_y, then compute P_0 as if you were using the `EXISTS` algebra. To emphasize that P_0 should be computed this way, we shall denote the value of t under P_0 as $\hat{\omega}^J_{P_0,\texttt{EXISTS}}(t)$. Since P is `COUNT(DISTINCT)`-safe, any proper ancestor P_0 of P_{-y} is of the form $P_0 = \pi^D_{-x} P_1$ or $P_0 = P_1 \bowtie P_2$. If $P_0 = \pi^D_{-x} P_1$ then $\hat{\omega}^J_{P_0}(t) = \coprod_{t' \in P_1} \hat{\omega}^J_{P_1}(t)$; this is correct because the tuples we are combining are disjoint, so which values are present does not matter. Else, we may assume $P_0 = P_1 \bowtie P_2$ and without loss we assume that $P_y \prec P_1$, thus we compute:

$$\hat{\omega}^J_{P_1,\texttt{COUNT(DISTINCT)}}(t) = \hat{\omega}^J_{P_1}(t_1) \otimes \hat{\omega}^J_{P_2,\texttt{EXISTS}}(t_2)$$

This is an abuse of notation since we intend that $\hat{\omega}^J_{P_2} \in \mathbb{Z}_2$ is first mapped into $\mathbb{Z}_{k+1}$ and then the convolution is performed. Since we are either multiplying our lossy vector by the annihilator or the multiplicative identity, this convolution has the effect of multiplying by the probability that t is in P_2, since these events are independent this is exactly the value of their conjunction. □

B.2.1 Complexity

Proposition B.2.2 (Second Half of Prop. 4.4.15)**.** *The following* `HAVING` *queries are* ♯P*-hard for* $i \geq 1$*:*

$$Q_{2,i}[\texttt{COUNT(DISTINCT}\, y) \,\theta\, k] \coloneq \mathtt{R}_1(x; y), \ldots, \mathtt{R}_i(x; y)$$

Proof. We start with $i = 1$. The hardness of $Q_{2,i}$ is shown by a reduction counting the number of set covers of size k. The input is a set of elements $U = \{u_1, \ldots, u_n\}$ and a family of sets $\mathcal{F} = \{S_1, \ldots, S_m\}$. A cover is a subset of $\mathcal{F}$ such that for each $u \in U$ there is $S \in \mathcal{S}$ such that $u \in S$. For each element $u \in U$, let $S_u = \{S \in \mathcal{F} \mid u \in S\}$, add a tuple $R(u; S; |S_u|^{-1})$ where $S \in S_u$. Every possible world corresponds to a set cover and hence, if W_k is the number of covers of size k then $\mu(Q) = W_k(\prod_{u \in U} |S_u|^{-1})$. Notice that if use the same reduction $i > 1$, we have that $\mu(Q) = W_k(\prod_{u \in U} |S_u|^{-i})$. □

We show that if Q contains self joins and is not `COUNT(DISTINCT)`-safe, then Q has ♯P data complexity. First, we observe a simple fact:

Proposition B.2.3. *Let* Q *be a* `HAVING` *query with an unsafe skeleton then* Q *has* ♯P*-hard data complexity. Further, if* Q *is connected and safe but not* `COUNT(DISTINCT)`*-safe then there must exist* $x \neq y$ *such that* $\forall g \in \mathbf{goal}(Q)$, $x \in \mathbf{key}(g)$.

Proof. We simply observe that the count of distinct variables is ≥ 1 exactly when the query is satisfied, which is ♯P-hard. The other aggregates follow easily. Since the skeleton of Q is safe, there is a safe plan for Q that is not `COUNT(DISTINCT)`-safe. This implies that there is some projection independent π^I_{-x} on all variables. □

Definition B.2.4. *For a conjunctive query* q*, let* F^q_∞ *be the least fixed point of* $F^q_0, F^q_1, \ldots$*, where*

$$F^q_0 = \{x \mid \exists g \in \mathbf{goal}(Q) \; s.t. \; \mathbf{key}(g) = \emptyset \;\wedge\; x \in \mathbf{var}(g)\}$$

and

$$F^q_{i+1} = \{x \mid \exists g \in \mathbf{goal}(Q) \; s.t. \; \mathbf{key}(g) \subseteq F_i \;\wedge\; x \in \mathbf{var}(g)\}$$

Intuitively, F^q_∞ is the set of variables "fixed" in a possible world.

Proposition B.2.5. *If q is safe and $x \in F_\infty^q$ then there is a safe plan P such that $\pi_{-x}^D \in P$ and for all ancestors of π_{-x}^D they are either $\pi_{-z}^D P_1$ for some z or $P_1 \bowtie P_2$.*

Proof. Consider the smallest query q such that the proposition fails where the order is given by number of subgoals then number of variables variables. Let $x_1, \ldots, x_n$ according to the partial order $x_i \prec x_j$ if exists F_k^q such that $x_i \in F_k^q$ but $x_j \notin F_k^q$. If $q = q_1 q_2$ such that $x \in \mathbf{var}(q_1)$ and $\mathbf{var}(q_1) \cap \mathbf{var}(q_2) = \emptyset$ then P_1 satisfies the claim and $P_1 \bowtie P_2$ is a safe plan. Otherwise let P_1 be a safe plan for $q[x_1 \to a]$ for some fresh constant a. Since this has fewer variables P_1 satisfies the claim and $\pi_{-x} P_1$ is safe immediately from the definition. □

We now define a set of rewrite rules $\Rightarrow$ which transform the skeleton and preserve hardness. We use these rewrite rules to show the following lemma:

Lemma B.2.6. *Let Q be a* `HAVING` *query using* `COUNT(DISTINCT)` *such that $q = \mathbf{sk}(Q)$ is safe, but Q is not* `COUNT(DISTINCT)`*-safe; and let there be some g such that $y \notin \mathbf{key}(g)$ and $y \notin F_\infty^q$ then Q has $\sharp\mathsf{P}$-hard data complexity.*

For notational convenience, we shall work with the skeleton of a `HAVING` query $Q[\alpha(y)\ \theta\ k]$ and assume that y is a distinguished variable.

1) $q \Rightarrow q[z \to c]$ if $z \in F_\infty^q$
2) $q \Rightarrow q_1$ if $q = q_1 q_2$ and $\mathbf{var}(q_1) \cap \mathbf{var}(q_2) = \emptyset$ and $y \in \mathbf{var}(q_1)$
3) $q \Rightarrow q[z \to x]$ if $x, z \in \mathbf{key}(g)$ and $z \neq y$
4) $q, g \Rightarrow q, g'$ if $\mathbf{key}(g) = \mathbf{key}(g')$, $\mathbf{var}(g) = \mathbf{var}(g')$ and $\mathbf{arity}(g) < \mathbf{arity}(g)'$
5) $q, g \Rightarrow q$ if $\mathbf{key}(g) = \mathbf{var}(g)$

We let $q \Rightarrow^* q'$ denote that q' is the result of any finite sequence of rewrite rules applied to q.

Proposition B.2.7. *If $q \Rightarrow^* q'$ and q' has $\sharp\mathsf{P}$-hard data complexity, then so does q.*

Proof. For rule 1, we can simply restrict to instances where $z \to c$. For rule 2, if q_1 is hard then q is hard because we can fill out each relation in q_2 with a single tuple and use q to answer q_1. Similarly, for rule 3 we can consider instances where $z = x$ so q will answer q_1. For rule 4, we apply the obvious mapping on instances (to the new subgoal). For rule 5, we fill out g with tuples of probability 1 and use this to answer q. □

Prop. B.2.6. By Prop. B.2.3, there is some x such that $x \in \mathbf{key}(g)$ for any $g \in \mathbf{goal}(Q)$. Let $q = \mathbf{sk}(Q)$, we apply rule 1 and 2 to a fixed point, which removes any products. We then apply the rule 3 as $\forall z \neq y$, $q[z \to x]$. Thus, all subgoals have two variables, x and y. We then apply rule 4 to a fixed point and finally rule 5 to a fixed point. It is easy to see that all remaining subgoals are of the form $R(x; y)$ which is the hard pattern. Further, it is easy to see that $g \Rightarrow^* R_i(x; y)$ for some i. □

We can now prove the main result:

Lemma B.2.8. *If Q is a* `HAVING` *query without self joins and Q is not* `COUNT(DISTINCT)`*-safe then the evaluation problem for Q is ♯P-hard.*

Proof. If q is unsafe, then Q has ♯P-hard data complexity. Thus, we may assume that q is safe but Q is not `COUNT(DISTINCT)`-safe. If Q contains $g \in \mathbf{goal}(Q)$ such that $y \in \mathbf{var}(g)$ but $y \notin \mathbf{key}(g)$ then Q has ♯P-hard data complexity by Lemma B.2.6. Thus, we may assume that y appears only in key positions.

First apply rewrite rule 2, to remove any products and so we may assume Q is connected. If Q is a connected and $y \in \mathbf{key}(g)$ for every g then Q is `COUNT(DISTINCT)`-safe. Thus, there are at least two subgoals and one contains a variable x distinct from y call them g and g' respectively. Apply the rewrite rule 3 as $q[z \to x]$ for each $z \in \mathbf{var}(q) - \{x, y\}$. Using rules 4 and 5, we can then drop all subgoals but g, g' to obtain the pattern $R(x), S(x, y)$, which is hard. □

B.3 Appendix: Full Proofs for `SUM` and `AVG`

B.3.1 `AVG` hardness

Definition B.3.1. *Given a set of nonnegative integers $a_1, \ldots, a_n$,*
the ♯`NONNEGATIVE SUBSET-AVG` problem is to count the number of non-empty subsets $S \subseteq 1, \ldots, n$ such that $\sum_{s \in S} a_s |S|^{-1} = B$ for some fixed integer B.

Proposition B.3.2. *♯`NONNEGATIVE SUBSET-AVG` is ♯P-hard.*

Proof. We first observe that if we allow arbitrary integers, then we can reduce any ♯`NONNEGATIVE SUBSET-SUM` with $B = 0$, which is ♯P-hard. Since the summation of any set is 0 if and only if their average is 0. Thus, we reduce from this unrestricted version of the problem. Let $B = \min_i a_i$ then

we simply make $a'_i = a_i + B$, now all values are positive, we then ask if the average is B. For any set S we have :

$$\sum_{s\in S} a'_s|S|^{-1} = \sum_{s\in S}(a_s + B)|S|^{-1} = \sum_{s\in S}(a_s + B)|S|^{-1} = \sum_{s\in S}|S|^{-1}a_s + B$$

Thus, it is clear that the average is satisfied only when $\sum_{s\in S} a_s = 0$. □

B.3.2 Proof of Theorem 4.4.21

It is sufficient to show the following lemma:

Lemma B.3.3. *Let $q = \mathbf{sk}(Q)$, if If q is safe, but Q is not SUM-safe then there is an instance I then for any set of values $y_1, \ldots, y_n$ let $q_i = q[y \to y_i]$ and $S \subseteq 1, \ldots, n$ we have $\mu(\bigwedge_{s\in S}^{n} q_s) = \prod_{s\in S}\mu(q_s) = 2^{-|S|}$. Further, on any world W and q_i there is a single valuation v for q_i such that $\mathbf{im}(q_i) \subseteq W$.*

Armed with his lemma we can always construct the distribution used in Prop. 4.4.20.

Proof. We observe that $y \notin F^q_\infty$ else there would be a SUM- and AVG-safe plan by Prop. B.2.5. Now consider the rewriting $q[x \to$ 'a'] for any $x \in F_\infty$ and $q[x \to y]$ if $x \notin F_\infty$. Thus, in any subgoal $y = \mathbf{var}(g)$. Pick one and add each y_1 value with probability $\frac{1}{2}$ independently. Notice that every relation either contains y_i in each tuple or the constant a. Since there are no self joins, this implies in any valuation either it must use a tuple containing y_i or the relation contains a single tuple with a for every attribute. Hence, the multiplicity of y_i is ≤ 1 in any possible world. Since there is only one relation with probabilistic tuples and all tuples have $\mu(t) = 0.5$, we have $\mu(\wedge_{s\in S} q_s) = 2^{-|S|}$ as required. □

Proposition B.3.4. *If $Q[\mathtt{SUM}(y) = k]$ is not SUM-safe and on a tuple independent instance, then Q does not have an* FPTRAS.

Proof. We observe that (SUM, =) is hard to approximate on even a single tuple-independent as a consequence of the previous reduction, which gives a one-to-one reduction showing (SUM, =) is as hard as

♯SUBSET-SUM, an NP-hard problem and so has no FPTRAS. □

B.4 Convergence Proof of Lemma 4.6.8

In the proof of this lemma, we need a technical proposition:

Proposition B.4.1. *Let q be a conjunctive query without self joins and R any relation contained in* **goal**(q)*, then $q(W,\tau) = \sum_{t\in R} q\left((W-R)\cup\{t\},\tau\right)$. Here, the summation is in the semiring S.*

Proof. By definition, the value of the query $q(W)$ can be written as $q(W,\tau) = \sum_{v\in\mathcal{V}}\prod_{g\in g}\tau(v(g))$. Since q does not contain self joins, each valuation contains exactly one member of R. Hence, there is a bijection between the between the two sums. Since semirings are associative, this completes the proof. □

We can now prove Lemma 4.6.8.

Lemma 4.6.8. We first observe that $W^O \subseteq W^R$, by Lemma 4.6.6, which shows $\frac{\mu(W^O)}{\mu(W^R)} \leq 1$. To see the other inequality, we construct a function $f : W^R \to W^O$ such that for any $W \in W^O$, $\frac{\mu(W)}{\mu(f^{-1}(W))} \geq (n+1)^{-1}\beta^{-1}$. This is sufficient to prove the claim. We describe f: if $W \in W^O$ then $f(W) = W$ else, $W \in W^R - W^O$ then we show that there is a tuple $t \in W$ such that $W - \{t\} \in W^O$, $f(W) = W - \{t\}$. Since there are at most n possible tuples to remove, this shows that $\left|f^{-1}(W)\right| \leq (n+1)$, Using the bounded odds equation, we have that $\frac{\mu(W)}{\mu(f^{-1}(W))} \geq (n+1)^{-1}\beta^{-1}$. Thus, all that remains to be shown is that we can always find such a tuple, t.

Consider $W \in W^R - W^O$, which means that $q(W,\tau^O) > k$ and $q(W,\tau^R) \leq n^2$. There must exist a tuple t such that $q(W,\tau^O) - q(W-\{t\},\tau^O) > k/n$ otherwise $q(W,\tau^O) \leq k$, which is a contradiction. To see this, consider any relation R in the query, we apply the above proposition to observe that:

$$\begin{aligned}\sum_{t\in R} q(W-\{t\},\tau^O) &= \sum_{t\in R}\sum_{s\in R:s\neq t} q(W-R\cup\{s\},\tau^O)\\ &= (|R|-1)\sum_{t\in R} q(W-R\cup\{t\},\tau^O)\\ &= (|R|-1)q(W,\tau^O)\end{aligned}$$

The second to last equality follows by counting how many times each term appears in the summation and that the semiring is embeddable in the rational numbers ($\mathbb{Q}$).

$$
\begin{aligned}
& q(W, \tau^O) - q(W - \{t\}, \tau^O) \leq k/n \\
\implies \quad & |R|\, q(W, \tau^O) + \sum_{t \in R} q(W - \{t\}, \tau^O) \leq k \\
\implies \quad & |R|\, q(W, \tau^O) + (|R| - 1) q(W, \tau^O) = q(W, \tau^O) \leq k
\end{aligned}
$$

The second line follows by summing over t in R, using the previous equation, and using that $|R| \leq n$. Thus, we can conclude there is some t such that $q(W, \tau^O) - q(W - \{t\}, \tau^O) > k/n$. By Lemma 4.6.6 we have:

$$q(W, \tau^R) \leq n^2 \implies \frac{n^2}{k} q(W, \tau^R) - \delta \leq n^2$$

Where $\delta \in [0, n)$. In turn, this implies

$$q(W, \tau^O) \leq \frac{k}{n^2}\delta + k \leq k + \frac{k}{n}$$

Since, $q(W, \tau^O) - q(W - \{t\}, \tau^O) > k/n$, we have that $q(W - \{t\}, \tau^O) \leq k$ and so $W - \{t\} \models Q^O$ and hence, $W - \{t\} \in W^O$. □

VITA

Christopher Ré was born in Los Angeles, California and raised in nearby Pasadena, California. He received a B.A. with a double major in Mathematics and Computer Science from Cornell University in 2001, an M.Eng in Computer Science in 2003 from Cornell University, and an M.S. in Computer Science in 2005 from the University of Washington. Starting in August of 2009, he will be an assistant professor at the University of Wisconsin–Madison.